2

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THREE ALGORITHMS FOR PLANAR-PATCH TERRAIN MODELING

by

Seung Hee Yee

June 1988

Thesis Advisor Neil C. Rowe

Approved for public release; distribution is unlimited.

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified		1b Restrictive Markings	
2a Security Classification Authority		3 Distribution Availability of Report Approved for public release; distribution is unlimited.	
2b Declassification Downgrading Schedule			
4 Performing Organization Report Number(s)		5 Monitoring Organization Report Number(s)	
6a Name of Performing Organization Naval Postgraduate School	6b Office Symbol *(if applicable)* 368	7a Name of Monitoring Organization Naval Postgraduate School	
6c Address *(city, state, and ZIP code)* Monterey, CA 93943-5000		7b Address *(city, state, and ZIP code)* Monterey, CA 93943-5000	
8a Name of Funding Sponsoring Organization	8b Office Symbol *(if applicable)*	9 Procurement Instrument Identification Number	
8c Address *(city, state, and ZIP code)*		10 Source of Funding Numbers Program Element No | Project No | Task No | Work Unit Accession No	
11 Title *(include security classification)* THREE ALGORITHMS FOR PLANAR-PATCH TERRAIN MODELING			
12 Personal Author(s) Seung Hee Yee			
13a Type of Report Master's Thesis	13b Time Covered From To	14 Date of Report *(year, month, day)* June 1988	15 Page Count 101
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
17 Cosati Codes Field | Group | Subgroup		18 Subject Terms *(continue on reverse if necessary and identify by block number)* Planar patch, Gradient, Curvature, Regions, Least-square fit	

19 Abstract *(continue on reverse if necessary and identify by block number)*

Providing a simplified model of real terrain has applications to route planning for robotic vehicles and military maneuvers. In this thesis I explore planar-patch surface modeling to represent terrain in a simple and effective way. In planar-patch surface modeling the terrain is subdivided into a set of planar subregions. The homogeneity of the gradient within a planar subregion simplifies calculating the cost of traversing the region, thus simplifying route planning. I have explored three main strategies to model the surface: joint top-down and and bottom-up, strict bottom-up, and presmoothing bottom-up approaches. Results of the algorithms are shown graphically by using the APL and Grafstat packages, verifying their correctness and accuracy.

20 Distribution Availability of Abstract ☒ unclassified unlimited ☐ same as report ☐ DTIC users	21 Abstract Security Classification Unclassified	
22a Name of Responsible Individual Neil C. Rowe	22b Telephone *(include Area code)* (408) 646-2158	22c Office Symbol 52Rp

DD FORM 1473,84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete

security classification of this page

Unclassified

Three Algorithms for Planar-Patch Terrain Modeling

by

Seung Hee Yee
Capt, Korean Army
B.S., Korea Military Academy, 1982

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1988

Author: [illegible]
Seung Hee Yee

Approved by: Neil C. Rowe
Neil C. Rowe, Thesis Advisor

[illegible]
Robert B. McGhee, Second Reader

[illegible]
Robert B. McGhee, Acting Chairman,
Department of Computer Sciences

[illegible]
James M. Fremgen,
Acting Dean of Information and Policy Sciences

ABSTRACT

Providing a simplified model of real terrain has applications to route planning for robotic vehicles and military maneuvers. ~~In~~ this thesis ~~I~~ explores planar-patch surface modeling to represent terrain in a simple and effective way. In planar-patch surface modeling the terrain is subdivided into a set of planar subregions. The homogeneity of the gradient within a planar subregion simplifies calculating the cost of traversing the region, thus simplifying route planning. ~~I have~~ explored three main strategies to model the surface: joint top-down and and bottom-up, strict bottom-up, and presmoothing bottom-up approaches. Results of the algorithms are shown graphically by using the APL and Grafstat packages, verifying their correctness and accuracy.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I. INTRODUCTION

A. BACKGROUND

There are many ways to represent a three-dimensional object, usually involving complicated mathematical and graphical theories. Representing and displaying a three-dimensional object like a terrain is important in applications such as cartography, route planning for robotic vehicles, road construction, and planning of military maneuvers. A simplified model of real terrain helps the user better understand and analyze it.

The simplest approximations are piecewise planar, and the very simplest is a polyhedron with triangular faces. Assume that a terrain is approximated by a two-dimensional array of evenly spaced elevations. Each group of three adjacent points in the array defines a planar polyhedral face, and a set of triplets constitutes a planar-patch approximation of the surface. Unfortunately, this approach creates many small triangular regions, which can make the searching process of route planning too complicated.

In this thesis, I developed three algorithms for planar-patch modeling of evenly spaced elevation data, and each has its own advantages and disadvantages. I have used the least-squares method to fit a plane over the points of a terrain, a quadtree algorithm to subdivide regions, a gradient clustering method to build regions, and smoothing algorithms on elevation data to improve continuity of adjoining regions.

One major advantage of planar-patch modeling over other three-dimensional surface modeling is that terrain within a patch is homogeneous in the sense of a constant gradient. This homogeneity simplifies calculating the cost or speed of traversing the region, thus making it easier to analyze for route planning.

Several thresholds affect my algorithms, and different threshold values will result in different models. These include the standard deviation of a fitted plane from given points, the ratio of magnitude between adjacent points, and the root mean square difference of coefficients of planar equations between adjacent regions. The final result of this thesis will be provided to Ron Ross (Ph.D. student, Computer Science Department, Naval Postgraduate School) for his research in route planning for robotic vehicles.

B. ORGANIZATION

Chapter 2 introduces terrain modeling techniques, including least-squares planar fitting and cell classification.

Chapte: ` is the main chapter of .his thesis, defining the planar-patch terrain model. Joint top-down and bottom-up and strict bottom-up planar-patch terrain modeling strategies are described in detail. Programming techniques such as the quadtree method used in the joint top-down and bottom-up algorithm and the region-growing method used in the strict bottom-up algorithm are explained. Finally, the issue of continuity of adjacent planar-patches along their common boundaries is discussed. Attempts to solve this problem, such as smoothing of original elevation data and grouping of the same type of points into one region are explained.

In Chapter 4 comparisons are made between the two algorithms in terms of accuracy and simplicity of terrain representation. Special features and the limitations inherent to each algorithm are discussed. Experimental results of joint top-down and bottom-up, strict bottom-up, and presmoothing bottom-up algorithms are presented in graphical form in this chapter. Pictures of the modeled terrain are shown in three dimensions using pictures drawn by the Grafstat and APL packages. Pictures of the boundaries of subregions are also included.

Finally, Chapter 5 summarizes the contributions made by this research and discusses some of the possible areas for further research based on this work. The Pascal source program is in the Appendix.

II. REVIEW OF TERRAIN MODELING

The most common method of terrain representation is by a set of functional expressions such as

$$f(x,y) = x^3 + 3x^2y - 5y + y^3$$

where f(x,y) is the elevation function for terrain and
x and y are the latitude and longitude coordinates.

This permits the simple calculation of the elevation given any combination of x and y. It is known that any continuous surface can be approximated with an arbitrarily small error by a polynomial of sufficiently high degree. Such polynomials can be derived by least-squares surface-fitting methods.

Surfaces may be defined in tabular form where the value of f is given for a selection of representative arguments. To find an arbitrary value of f, a table lookup can be performed to determine the value of the nearest known points and use them to approximate the desired value by interpolation. [Ref. 1: p. 212]

It is possible to specify a surface in terms of guiding points by a generalization of the Bezier polynomials or the B-splines. The Bezier polynomial or B-spline method finds an approximate curve that passes near a set of given points. [Ref. 2: p. 247]

A. TERRAIN CELL CLASSIFICATION

1. QUADRATIC APPROXIMATION OF SURFACE BY LEAST SQUARES FIT

Fitting a least-squares quadratic to a 3 by 3 square of points on a surface is done in the following way. The fit as a function of x and y is: [Ref. 3: p. 55]

$$\hat{f}(x,y) = k_1 + k_2x + k_3y + k_4x^2 + k_5xy + k_6y^2$$

The square of the error between the fit equation and the given terrain points is:

$$E^2 = \sum_x \sum_y (k_1 + k_2x + k_3y + k_4x^2 k_5xy + k_6y^2 - f(x,y))^2$$

The partial derivatives of the squared error with respect to each constant are:

$$\frac{\partial E^2}{\partial k_1} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]1 \tag{5}$$

$$\frac{\partial E^2}{\partial k_2} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]x \tag{6}$$

$$\frac{\partial E^2}{\partial k_3} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]y \tag{7}$$

$$\frac{\partial E^2}{\partial k_4} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]x^2 \tag{8}$$

$$\frac{\partial E^2}{\partial k_5} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]xy \tag{9}$$

$$\frac{\partial E^2}{\partial k_6} = \left[2\sum_x\sum_y(k_1 + k_2x + k_3y + k_4x^2k_5xy + k_6y^2 - f(x,y))\right]y^2 \tag{10}$$

To minimize the squared error, each of the above equations is set to zero and simplified. Those equations can be further simplified by assuming that the coordinates of each of the points in the 3 by 3 matrix, except the center point, is $\pm$ one distance unit from the center point. By the above assumption, any terms in above equations which contain a summation of an x or y coordinate with an odd power will go to zero. Simplified equations are:

$$0 = k_1\sum_x\sum_y + k_4 \times 6 + k_6 \times 6 - \sum_x\sum_y f(x,y) \tag{11}$$

$$0 = k_2 \times 6 + \sum_x\sum_y f(x,y)x \tag{12}$$

$$0 = k_3 \times 6 + \sum_x\sum_y f(x,y)y \tag{13}$$

$$0 = k_1 \times 6 + k_4 \times 6 + k_6 \times 4 - \sum_x \sum_y f(x,y)x^2 \qquad (14)$$

$$0 = k_5 \times 4 - \sum_x \sum_y f(x,y)xy \qquad (15)$$

$$0 = k_1 \times 6 + k_4 \times 4 + k_6 \times 6 - \sum_x \sum_y f(x,y)y^2 \qquad (16)$$

From above equations, if we solve for k_1, k_2, k_3, k_4, k_5 and k_6, we get a quadratic equation fitted to the 3 by 3 square of points.

2. TERRAIN CELL CLASSIFICATION BASED ON QUADRATIC APPROXIMATIONS

The eigenvalues of the Hessian matrix for the quadratic function approximated for the 3 by 3 points on the surface are: [Ref. 3: p. 61]

$$\lambda = \frac{(2k_4 + 2k_6) \pm \sqrt{(-2k_4 - 2k_6)^2 - 4(2k_4 2k_6 - k_5^2)}}{2}$$

Using the signs of the eigenvalues, we can classify the central point by Table 1 on page 6. [Ref. 3: p. 69]

Table 1. **CLASSIFICATION OF SURFACE POINTS BY EIGENVALUES OF THE HESSIAN MATRIX**

sign of λ_1	sign of λ_2		
	-	0	+
+	saddle	impossible	depression
0	ridge	plane	valley
-	hill	impossible	pass

B. PLANE APPROXIMATION OF A SURFACE BY LEAST SQUARES FIT

The elevation z of any point (x,y) can be expressed as:

$$z = f(x,y)$$

The plane that we want to fit over the surface is:

$$z = ax + by + c$$

The square of the error between the fitted plane and the real terrain is:

$$E^2 = \sum_x \sum_y (ax + by + c - f(x,y))^2$$

The partial derivatives of the squared error with respect to a,b and c are:

$$\frac{\partial E^2}{\partial a} = \left[2 \sum_x \sum_y (ax + by + c - f(x,y)) \right] x \tag{21}$$

$$\frac{\partial E^2}{\partial b} = \left[2\sum_x\sum_y (ax + by + c - f(x,y)) \right] y \tag{22}$$

$$\frac{\partial E^2}{\partial c} = \left[2\sum_x\sum_y (ax + by + c - f(x,y)) \right] 1 \tag{23}$$

To minimize the squared error, each of the above equations is expanded and then the partials are set to zero.

$$0 = \sum_x\sum_y ax^2 + \sum_x\sum_y bxy + \sum_x\sum_y cx - \sum_x\sum_y f(x,y)x \tag{24}$$

$$0 = \sum_x\sum_y axy + \sum_x\sum_y by^2 + \sum_x\sum_y cy - \sum_x\sum_y f(x,y)y \tag{25}$$

$$0 = \sum_x\sum_y ax + \sum_x\sum_y by + \sum_x\sum_y c - \sum_x\sum_y f(x,y) \tag{26}$$

Here, we have three unknowns, a,b and c, and three equations. So we can solve for them.

C. THE GRADIENT AT A POINT

One of the primary criterion we will use for grouping points into a region is the gradient. The gradient of a point is a vector that has a magnitude (slope) and a direction for its components. We can represent a three by three elevation matrix of terrain points as in Table 2 on page 8, where x and y are the coordinates of a point on a terrain and f(x,y) is the elevation of the point.

Table 2. A MATRIX OF TERRAIN POINTS

f(x-1,y+1)	f(x,y+1)	f(x+1,y+1)
f(x-1,y)	f(x,y)	f(x+1,y)
f(x-1,y-1)	f(x,y-1)	f(x+1,y-1)

The gradient vector of a point (x,y) can be approximated in three ways. The first method is:

$$\Delta_1 = f(x+1,y) - f(x,y)$$

$$\Delta_2 = f(x,y+1) - f(x,y)$$

$$magnitude(x,y) = \sqrt{(\Delta_1^2 + \Delta_2^2)}$$

$$direction(x,y) = \tan^{-1}(\frac{\Delta_2}{\Delta_1})$$

The second method is:

$$\Delta_1 = f(x+1,y) - f(x-1,y)$$

$$\Delta_2 = f(x,y+1) - f(x,y-1)$$

The magnitude(x,y) and direction(x,y) are the same as above.

The third way of approximating gradient is by first fitting a quadratic to the three by three square of points and then using the coefficients of the quadratic : [Ref. 3: p. 60]

$$magnitude(x,y) = \sqrt{(k_2^2 + k_3^2)}$$

$$direction(x,y) = \tan^{-1}(\frac{k_3}{k_2})$$

where k_2 and k_3 are the coefficients of the quadratic described in section A.

The first method considers only one quadrant of the 3 by 3 matrix, thus is biased. The second method uses a point from each of the quadrants, and is better than the first method in the sense that it utilizes more unbiased information. However, it gives an incorrect value in the case as shown in Table 3.

Table 3. AN EXAMPLE OF A TERRAIN POINTS MATRIX

5	2	3
3	1	3
3	2	4

When we calculate the gradient of center point using the second method, it will give the magnitude value zero and direction value 45 degree, which is wrong. The third method does not have such problem, since it utilizes all eight neighboring points. In this case it gives the magnitude value 0.2357 and direction value 45 degree. I used the third method in my program.

The gradient vector of a terrain point points in the x-y direction of maximum slope of that point; *magnitude(x,y)* is the maximum slope value and *direction(x,y)* is the direction of that slope in the x-y plane.

III. PLANAR-PATCH TERRAIN MODELS

A planar-patch terrain model is a model that represents terrain in the form of a polyhedron. This model is different from the common planar terrain model used in graphics in that size of a planar-patch is not fixed, and the shape of a patch does not have to be regular.

A. IMPLEMENTATION

I chose Pascal as an implementation language for this research because of my familiarity and experience, and the Waterloo Pascal on the IBM 370/3033AP mainframe was the Pascal compiler that was used.

The APL language and the Grafstat graphics software package were used to verify the correctness and accuracy of the algorithms. Even though I chose Pascal as my implementation language, the matrix form of the elevation data suggests APL. With the Grafstat software package which runs in the APL environment, one can check the intermediate results of an APL program by displaying pictures. Since I experimented with different threshold values in my algorithm, APL and Grafstat were useful. The three-dimensional pictures in chapter 4 were done with Grafstat.

Two arrays of records are used as main data structures by all my algorithms: **point_array** and **subregion_array**. The **point_array** is a two-dimensional array which corresponds to the points in the real terrain. Each element of the array is a record which has the elevation, the subregion ID, and the magnitude of the terrain point as its fields. From the values of elevation and magnitude, the subregion ID is determined and assigned to the point. The **subregion_array** is a one-dimensional array of records which is created during the subregion creation phase. The indices of this array are the ID's of subregions. It keeps necessary information about the subregions such as the equation of the plane fitted to the subregion, the boundary of the subregion, and other subsidiary information.

The general structure of my first two modeling algorithms is shown in Figure 1 on page 12.

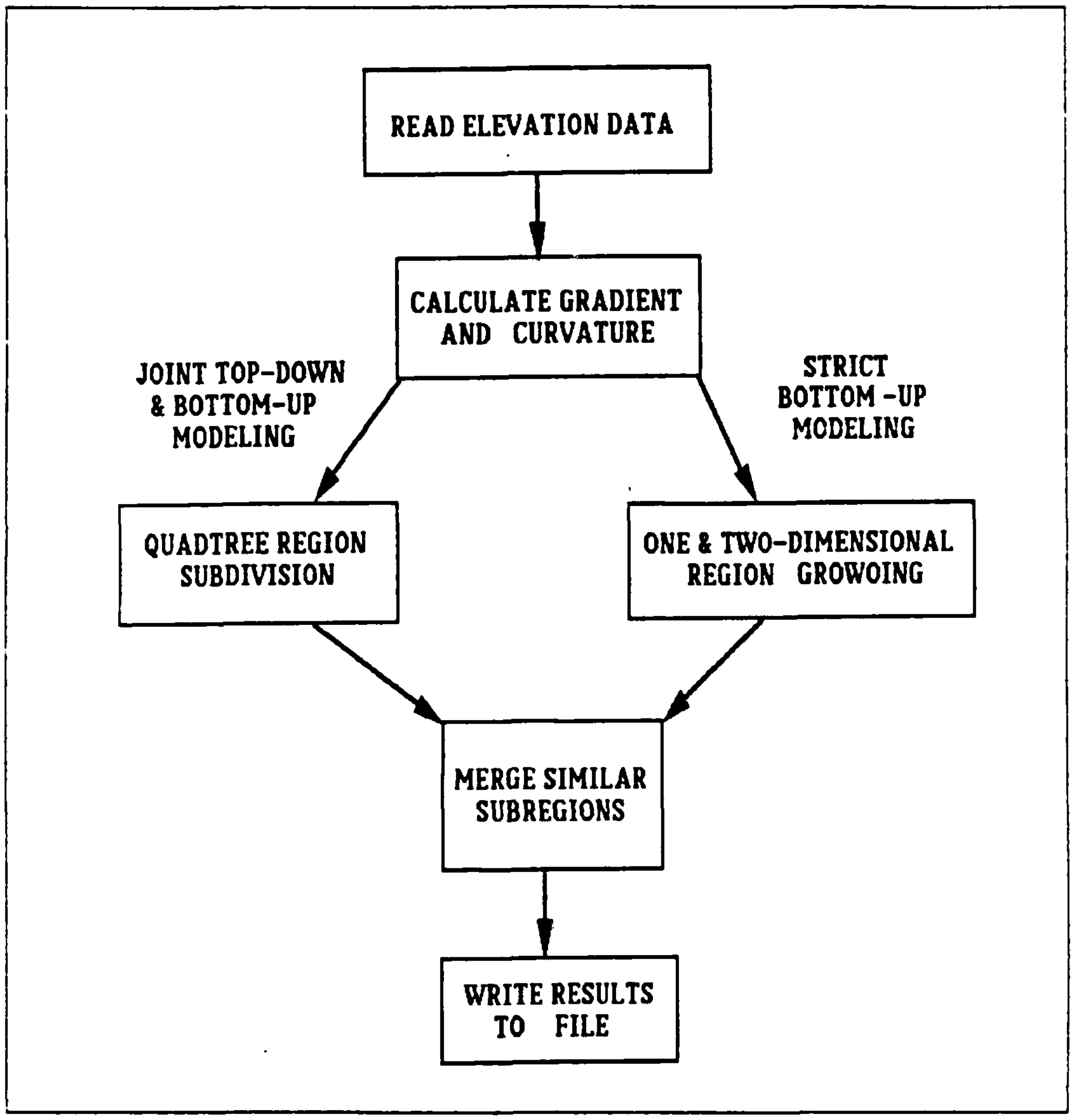

READ ELEVATION DATA
CALCULATE GRADIENT AND CURVATURE
JOINT TOP-DOWN & BOTTOM-UP MODELING
STRICT BOTTOM -UP MODELING
QUADTREE REGION SUBDIVISION
ONE & TWO-DIMENSIONAL REGION GROWOING
MERGE SIMILAR SUBREGIONS
WRITE RESULTS TO FILE

Figure 1. Block Diagram of the First Two Program Approaches

B. JOINT TOP-DOWN AND BOTTOM-UP PLANAR-PATCH TERRAIN MODELING

My first algorithm consists of two parts. The first part is the top-down subdivision of the original elevation data matrix into subregions and the second part is the bottom-up merging of adjoining subregions when adjoining subregions have similar plane equations.

1. TOP-DOWN PHASE

A quadtree region subdivision method is used during the top-down phase of the firtst algorithm. The method generates a quadtree by recursively dividing a two-dimensional region into quadrants. Each node in the quadtree has four data elements, one for each of the quadrants in the region. [Ref. 4: p. 215] The original region is given some standard tests such as comparison to a threshold of the maximum magnitude or the minimum magnitude of the points within it, depending on the application. Only if a region fails the tests and is not of minimum size, it is further subdivided into four subregions. If a region has both length and width four cells or more, it is not of the minimum size. See Figure 4 on page 16 for the different kinds of minimal subregions. The subdivision process described above is repeated for each subregion. Figure 2 on page 14 shows graphically this algorithm.

The code works whether a region has a dimensions of odd or even numbers. For example, consider 4 by 4 and 5 by 5 matrix regions to subdivide; Figure 4 on page 16 shows how this method works.

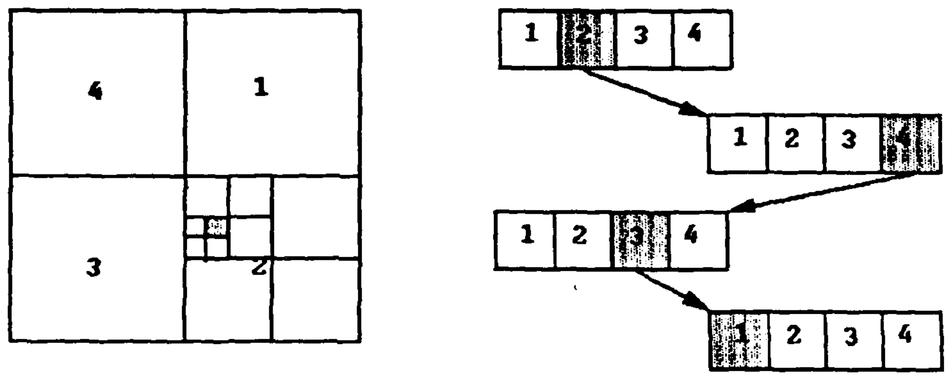
2. THRESHOLDS USED

Three thresholds are used in the top-down phase: **low_bound**, **upper_bound** and **standard_deviation (SD)**.

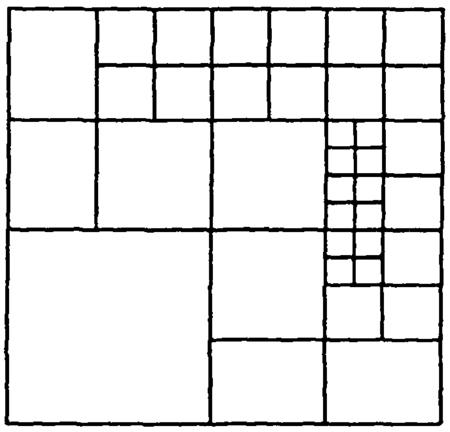
The threshold **upper_bound** is set by the maximum climbing capability of the vehicle in cases where the polyhedral model will be used for route planning. If the absolute value of the minimum slope of a region is greater than the **upper_bound**, which means every cell has a slope greater than the **upper_bound**, then the region is assumed being untraversable by the vehicle, and the region need not be subdivided further because a good polyhedral fit just doesn't matter for that region. The threshold **low_bound** means that when the absolute value of the maximum slope of a region is less than the **low_bound**, which means every cell has a slope less than the **low_bound**, then the slopes in the region are unimportant. Such regions need not be subdivided either.

For the region which has maximum slope greater than the **low_bound** and minimum slope less than the **upper_bound**, the region is subdivided in four parts according to the quadtree algorithm to better isolate the unlevel slopes. Then we fit a plane to the region and calculate the standard deviation of the plane from the real terrain points. The standard deviation (SD) is:

$$SD = \sqrt{\sum_x \sum_y \frac{(f(x,y) - (ax + by + c))^2}{n - 1}}$$



QUADTREE REGION SUBDIVISION METHOD



AN EXAMPLE OF SUBREGIONS CREATED BY THE QUADTREE METHOD

Figure 2. Quadtree Region Subdivision Method

```
PROCEDURE MAKE_SUBREGIONS(YL,YU,XL,XU : INTEGER;
                          prev_region : region_ptr);

(* YL : LOWER BOUND IN Y AXIS, YU : UPPER BOUND IN Y AXIS,
   XL : LOWER BOUND IN X AXIS, XU : UPPER BOUND IN X AXIS *)

VAR
   P : REGION_PTR;

BEGIN

  IF ((YU-YL) > 2) AND ((XU-XL) > 2) THEN (* NOT MINIMUM SIZE *)

     if (p@.max_mag_of_region > low_bound) then    (* test *)
     begin
       make_subregions(((yu-yl) div 2)+yl+1 , yu,
         ((xu-xl) div 2)+xl+1 , xu , p);    (* 1st quadrant *)
       make_subregions(yl , ((yu-yl) div 2)+yl,
         ((xu-xl) div 2)+xl+1 , xu , p);    (* 2nd quadrant *)
       make_subregions(yl , ((yu-yl) div 2)+yl,
          xl , ((xu-xl) div 2)+xl, p);      (* 3rd quadrant *)
       make_subregions(((yu-yl) div 2)+yl+1 , yu,
          xl , ((xu-xl) div 2)+xl , p);     (* 4th quadrant *)

        p := prev_region ;            (* goto parent region *)
     end
     else stop_subdivide  (* region passed standard test *)
  else stop_subdivide     (* region has minimum size     *)

END
```

Figure 3.   Pascal Code for the Quadtree Region Subdivision Method

where $f(x,y)$ is the elevation of point (x,y)

$ax + by + c$ is the plane equation of the region

n is the total number of points in that region.

The region is subdivided only if the calculated SD is bigger than the threshold SD and it does not have the minimum size.

3. BOTTOM-UP PHASE

We now merge adjoining regions having similar plane expressions. The reason is that in the process of applying the quadtree subdivision, we often subdivide a region which should not have been so completely subdivided. For example, say a region has very steep slopes in the first quadrant and the rest of the quadrants are flat. Since the original region will fail in the maximum slope test due to the slopes in the first quadrant,

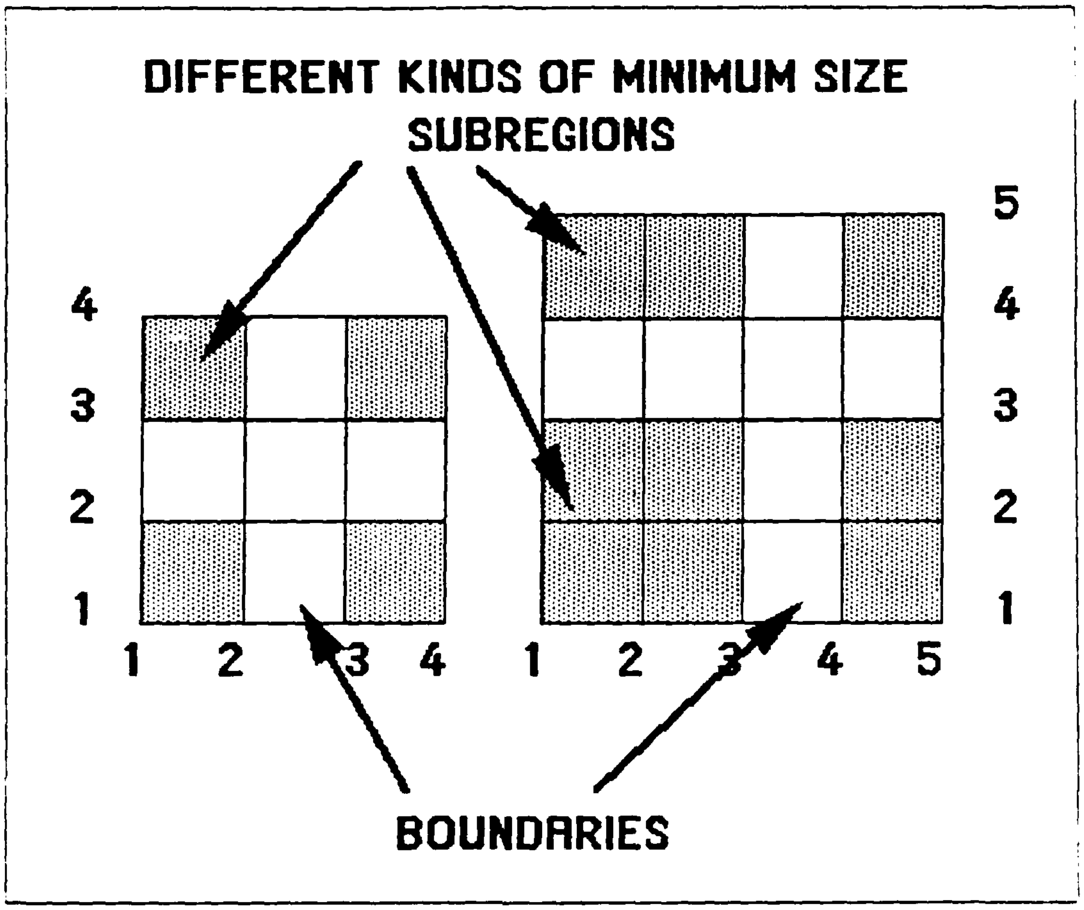


Figure 4. Minimum-Size Subregions Created by the Quadtree Method: This picture shows quadtree subdivision method applied to the regions which have dimensions of odd or even numbers, and also shows boundaries and different kinds of minimum size subregions.

the top-down phase will subdivide it in four. However, the first, second and third quadrants are all flat and should not be separate.

The adjoining regions of a region are found from the **point_array** by looking up the subregion ID field of adjoining point. The bottom-up phase is done in two subphases: **row_combine** and **col_combine**. The **row_combine** finds and merges adjoining regions which are located in a same row, and the **col_comnine** finds and merges the regions in a same column.

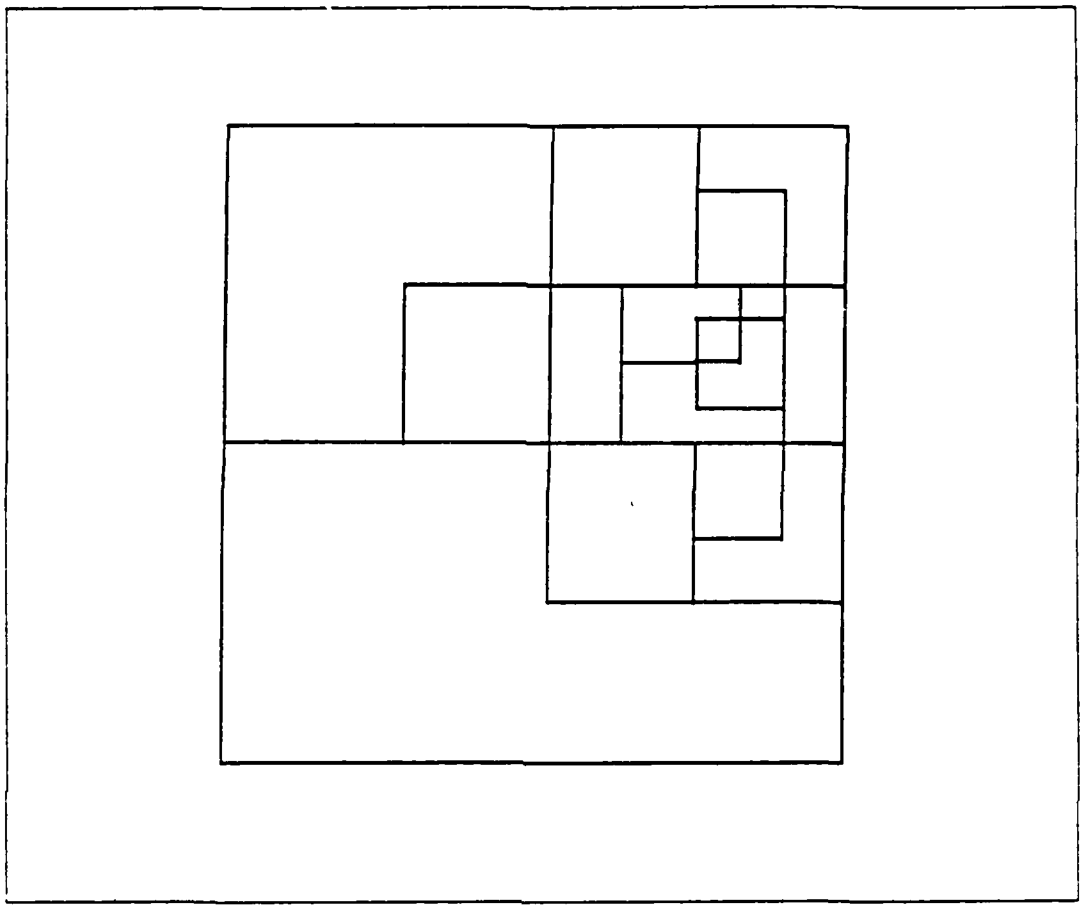


Figure 5. **Subregions after the Merging Process:** These subregions are created by performing the merging process over the initial subregions created by the quadtree algorithm.

To merge adjoining subregions which have similar plane equations, a value **equation_difference** is calculated from the two plane equations and compared to the threshold **abc_difference**. The **equation_difference** is:

$$equation\_difference = \sqrt{(a_1 - a_2)^2 + (b_1 - b_2)^2 + (c_1 - c_2)^2}$$

where a_1, b_1 and c_1 are the coefficients of $plane_1$ and

a_2, b_2 and c_2 are the coefficients of $plane_2$.

If this is less than the threshold then the two adjoining regions pass the test. Then we fit a plane over the two subregions and calculate the standard deviation of the plane, and only if it is less than the threshold **standard_deviation** we do merge the two subregions.

C. STRICT BOTTOM-UP PLANAR-PATCH TERRAIN MODELING

1. POINT-CLASSIFICATION PHASE

In strict bottom-up modeling, we first classify every point in a region by attaching two tags to it. The first tag classifies slope of the point: level, safe-slope or unsafe-slope. A threshold **low_bound** defines the low limit of slope and another threshold **upper_bound** defines the upper limit. The second tag classifies the curvature of the point, by the terrain cell classification criteria described in chapter 2. The designators are hill, depression, plane and special. The tag special refers to all the rest of the cell classifications except the above three. A threshold called **curvature** is compared to the eigenvalue of the Hessian matrix; if its absolute value is less than the threshold, then the eigenvalue is regarded as zero.[Ref. 3: p. 69] We group the contuguous points with the same first and second tag to form subregions.

Next the thresholds **magnitude_ratio** and **abc_difference** are used to combine the subregions. The **magnitude_ratio**, is used in the **region-growing** phase and the **abc_difference** is used as in joint top-down and mottom-up modeling.

2. REGION-GROWING PHASE

The region-growing phase consists of two subphases: one-dimensional and two-dimensional.

a. ONE-DIMENSIONAL REGION-GROWING

The one-dimensional phase starts with the first point (p_{11}) in the first row of the **point_array**. The point becomes the head of a linked list and the **subregion_ID** field of the point is given the value one. The subregion ID starts from one and is incremented every time a new subregion is created. Whenever a new subregion is created, the necessary information about the subregion, such as the coordinates of the starting point and boolean value that says whether the subregion is active or not, is stored in the **subregion_array** using the subregion ID as an index. Then it examines the next point (p_{12}) in the same row. If the ratio computed using magnitudes of gradient ($\frac{abs(magnitude_{p12} - magnitude_{p11})}{magnitude_{p11}}$) is less than the threshold **magnitude_ratio**, then they are linked together to form a linear subregion (R_{11}). That is, the **next_point** field of the first point in the **point_array** is given the x and y coordinate of the second point and the **subregion_ID** field of the second point is given the same value as the subregion ID of the first point. The average magnitude of the gradient of the points within the subregion is

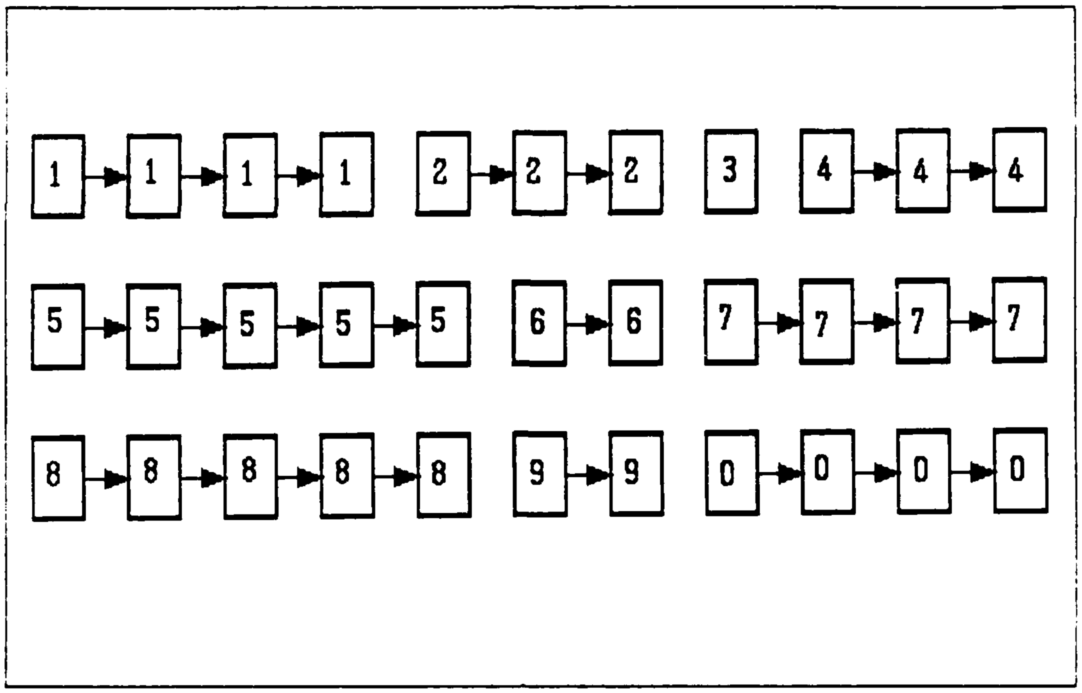


Figure 6. One-Dimensional Region-Growing

calculated and is stored in the **avg_mag** field of the subregion record in the **subregion_array**. Then repeatedly for each next point (p_{1k}) in the same row, the ratio between the average magnitude of the gradient of the subregion and the magnitude of the gradient of the new point is compared to **magnitude_ratio**. If on this basis new point (p_{1k}) should not join the subregion, a new region starts from that new point and the new point becomes a head of another linked list. This process continues for all rows. See Figure 6 for an example of this algorithm.

b. *TWO-DIMENSIONAL REGION-GROWING*

The two-dimensional subphase starts with the first linear subregion (R_{11}) which was created by the one-dimensional subphase. Actually, it starts from the first point in the first column of the **point_array**, which belongs to the subregion one. It examines the point below in the same column, which belongs to another subregion, to decide whether the average magnitude of the gradient of the subregions are similar. The information, average magnitudes of the gradient of the two subregions, is retrieved from the the **subregion_array** using the subregion ID of each point. If the ratio is less than **magnitude_ratio** then they are linked together. The tail of the first subregion is linked

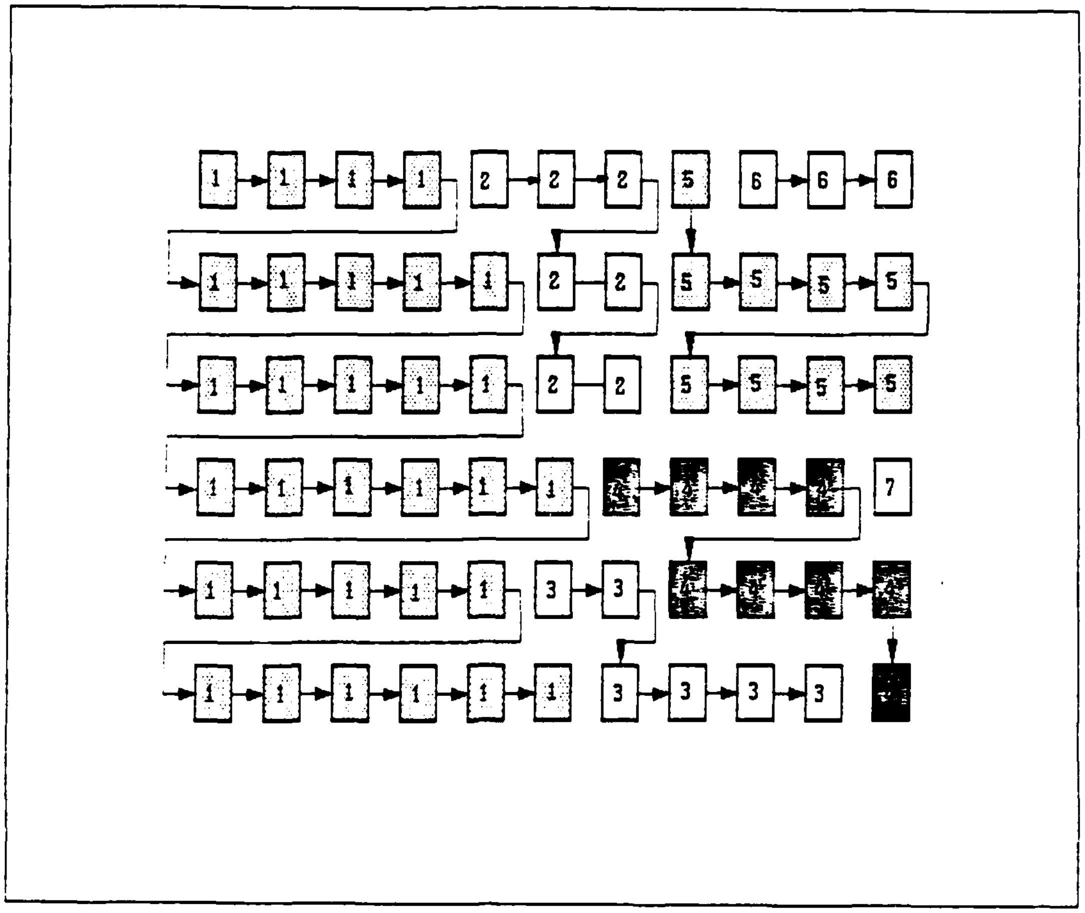


Figure 7. **Two-Dimensional Region-Growing:** This algorithm is applied after the one-dimensional region-growing algorithm.

to the head of the second subregion, the tail of the second subregion is left nil, all the subregion IDs of the points within the second subregion are changed to the first subregion ID, and the second subregion in the **subregion_array** is marked inactive. When those two linear subregions are merged, the resulting subregion is not one-dimensional any more, except if both subregions were single points. Then repeatedly for each next point in the same column, the ratio between the average magnitudes of the gradient of the subregions is compared to the **magnitude_ratio**. If a new subregion does not join the old subregion, the control is moved to the new subregion and the process starts from that new subregion. This process continues for all columns. This is illustrated in Figure 7. After the two-dimensional phase, we fit a plane to each subregion using least-squares. Then the threshold **abc_difference** is used to merge adjoining regions that have similar plane expressions. The merging is done in two steps. The first step is to scan each row of the **point_array** looking for adjacent points in the same row which have different subregion IDs. For every pair of adjacent subregions, the plane expressions are consulted from the **subregion_array** by using the ID numbers of the subregions as the indices. The **equation_difference** of the two plane equations are calculated and compared to **abc_difference**. If it less, the two subregions are merged. point of the second subregion to the tail point of the first subregion. The second step is identical except that scanning for merging is done by column.

D. AN ALGORITHM IMPROVING THE CONTINUITY OF ADJOINING REGIONS

One common problem with planar-patch terrain modeling is that planar patches for two adjacent subregions seldom yield a surface that is continuous along the line segments bisecting the points used to create the regions. Since the initial requirements for this research rule out any other terrain modeling than planar-patch, we had to come up with some idea to ensure continuity of the patches. One simple solution is to model terrain entirely with small triangles, each of which exactly fits three adjacent terrain points, as described in Chapter 1. The number of triangles necessary is very large.

But a minor modification to the two modeling approaches explored in this research can give smoothness. One can create supplementary triangular subregions to fill the gap between adjoining subregions. The gap between adjoining subregions have two lines, one for each side of the gap. The lines are the edges of the adjacent planes. We can define two triangles on the gap by grouping the first end point of the first edge to the second edge and second point of the second edge to the first edge. See Figure 8 on page 22.

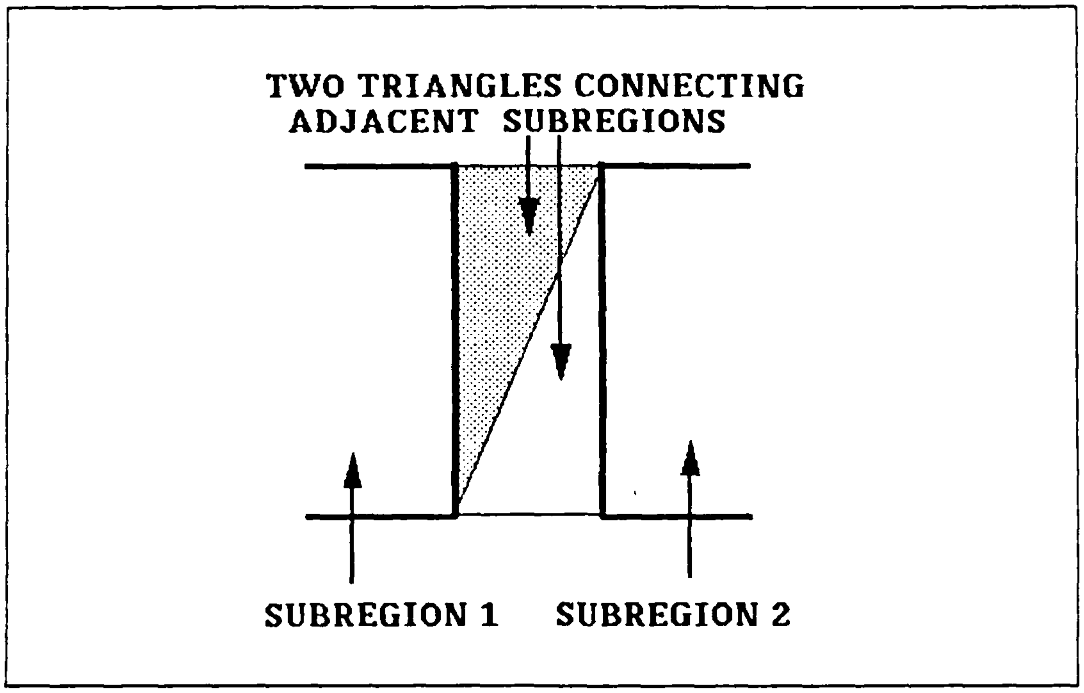


Figure 8. Filling the Boundary with Two Triangles

This is better than the many-triangle model, but is still not optimal, because the number of subregions will be increased considerably by the supplementary triangular subregions.

We have made a different attempt to solve this problem using smoothing of the elevation array and an expansion to three dimensions of Rolle's theorem. The basic idea is that if we fit planar-patches to an everywhere-convex or everywhere-concave region, the fitted patches are much more likely to be continuous. This means we should try to ensure that all the points within large regions are classified as either planar points, hill points or depression points. See Figure 9 on page 23. Smoothing the elevation data will help: other kinds of points will tend to be changed into one of the three.

Unfortunately, because of time, we were not able to solve this rather complex problem completely. It is recommended for future research.

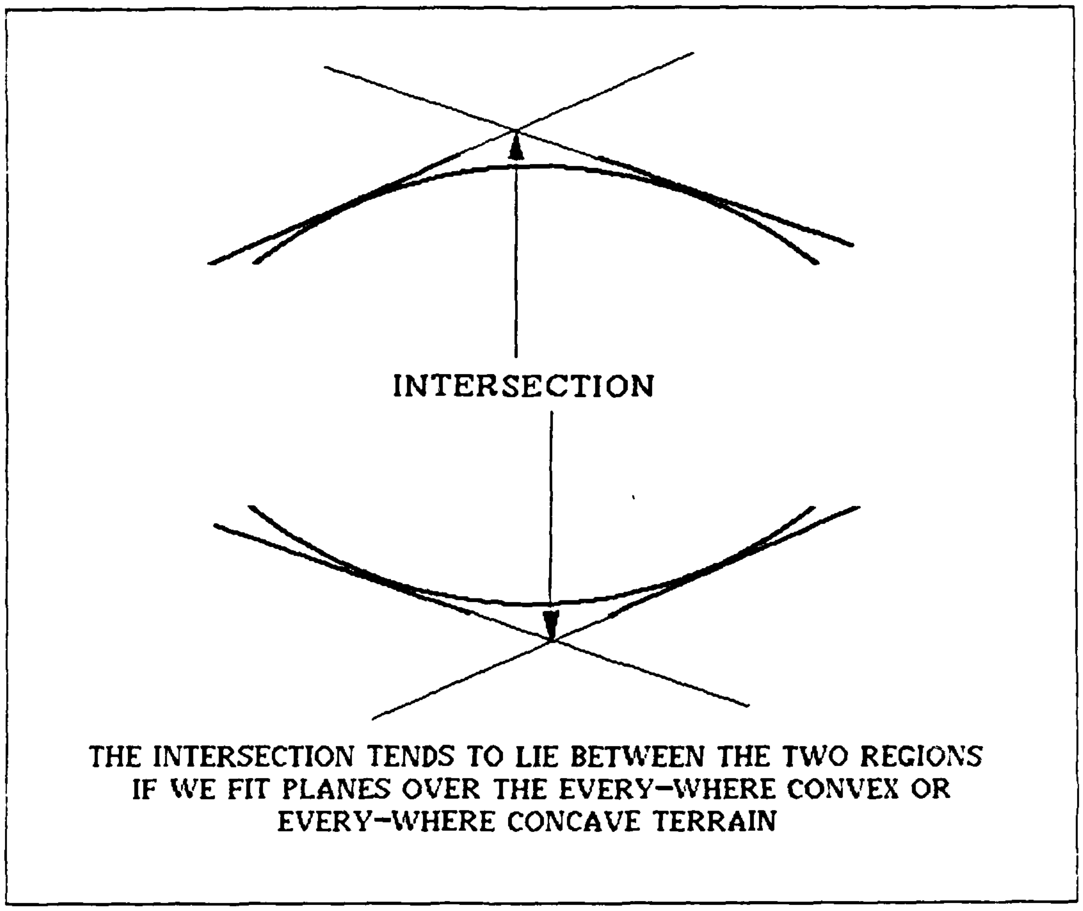

INTERSECTION
THE INTERSECTION TENDS TO LIE BETWEEN THE TWO REGIONS
IF WE FIT PLANES OVER THE EVERY-WHERE CONVEX OR
EVERY-WHERE CONCAVE TERRAIN

Figure 9. How Surface Convexity Helps

IV. DISCUSSION

A. ACCURACY ANALYSIS

The joint top-down and bottom-up and strict bottom-up approaches produce fairly good terrain models in terms of accuracy and simplicity. Here accuracy means whether the model represented the real terrain without losing anything significant, and simplicity means the number of subregions.

The overall accuracy of a model can be verified by comparing the pictures of the real terrain and model. But there is another thing that describes the accuracy of a model, standard deviation. We can evaluate the local accuracy in terms of the standard deviation of the elevation difference for a point in the real terrain and the corresponding point in the model. In the top-down approach, each subregion must have a fit standard deviation less than the the threshold **standard_deviation**. Let's assume that the elevation difference between a point on the real terrain and a point on the model has a normal probability distribution. If we know the standard deviation of the elevation differences of a subregion:

1. About 68 percent of the points will have elevation difference less than plus or minus one standard deviation.
2. About 98 percent of the points will have elevation difference less than plus or minus two standard deviations.
3. Practically all points (99.73 percent) will have elevation difference less than plus or minus three standard deviations.

So for example, if we limit the model to have the worst case local elevation difference, say 10 feet, with 98 percent certainty, we can set the standard deviation threshold to 5 feet. Consequently every subregion will have a standard deviation less than 5 feet, and 98 percent of the points will have an elevation difference less than 10 feet.

The threshold standard deviation was not used in the strict bottom-up approach so the above argument does not hold.

B. LIMITATIONS OF THE MODEL

In the joint top-down and bottom-up approach, the minimum number of points in a subregion is four and the shape of initial subregion must be a rectangle. This restriction is due to the quadtree. These limitations prohibit the approach from picking up linear (one-cell-wide) terrain features and creating linear subregions. This limitation, however, does not exist in the strict bottom-up approach owing to the linked list of points used to represent a region. When a linear region is a straight horizontal or vertical line, one cannot calculate a plane equation.

C. EXPERIMENTAL RESULTS

We now present the output of our terrain modeling program in graphical form. It is applied to sample elevation data acquired from the Fort Hunter Liggett terrain data base. Shown are the pictures that are produced by our modeling programs using different thresholds.

As was mentioned in chapter 3, in joint top-down and bottom-up modeling, the thresholds were **low_bound** and **upper_bound** for gradient magnitude, **standard_deviation,** and **abc_difference**. In strict bottom-up modeling, the thresholds were **low_bound** and **upper_bound** of gradient magnitude, **curvature**, **abc_difference**, and **magnitude_ratio**.

Three sets of pictures are given sequentially in the following section. The first set is produced by the joint top-down and bottom-up approach (referred as the first approach in the following pictures), the next by the strict bottom-up approach (referred as the second approach), and the last by the bottom-up approach applied to the smoothed data (referred as the third approach).

In the array pictures, each subregion has a unique code. If a number does not have a prefix then the number is actually the subregion ID. If it does have a prefix then the number has to be converted. See the conversion example below.

23	= = >	subregion ID 23.
.23	= = >	subregion ID 123.
:23	= = >	subregion ID 223.
-23	= = >	subregion ID 323.
0	= = >	either an outermost boundary or inside area of a region.

1. EXPERIMENTAL RESULTS 1 OF THE FIRST APPROACH

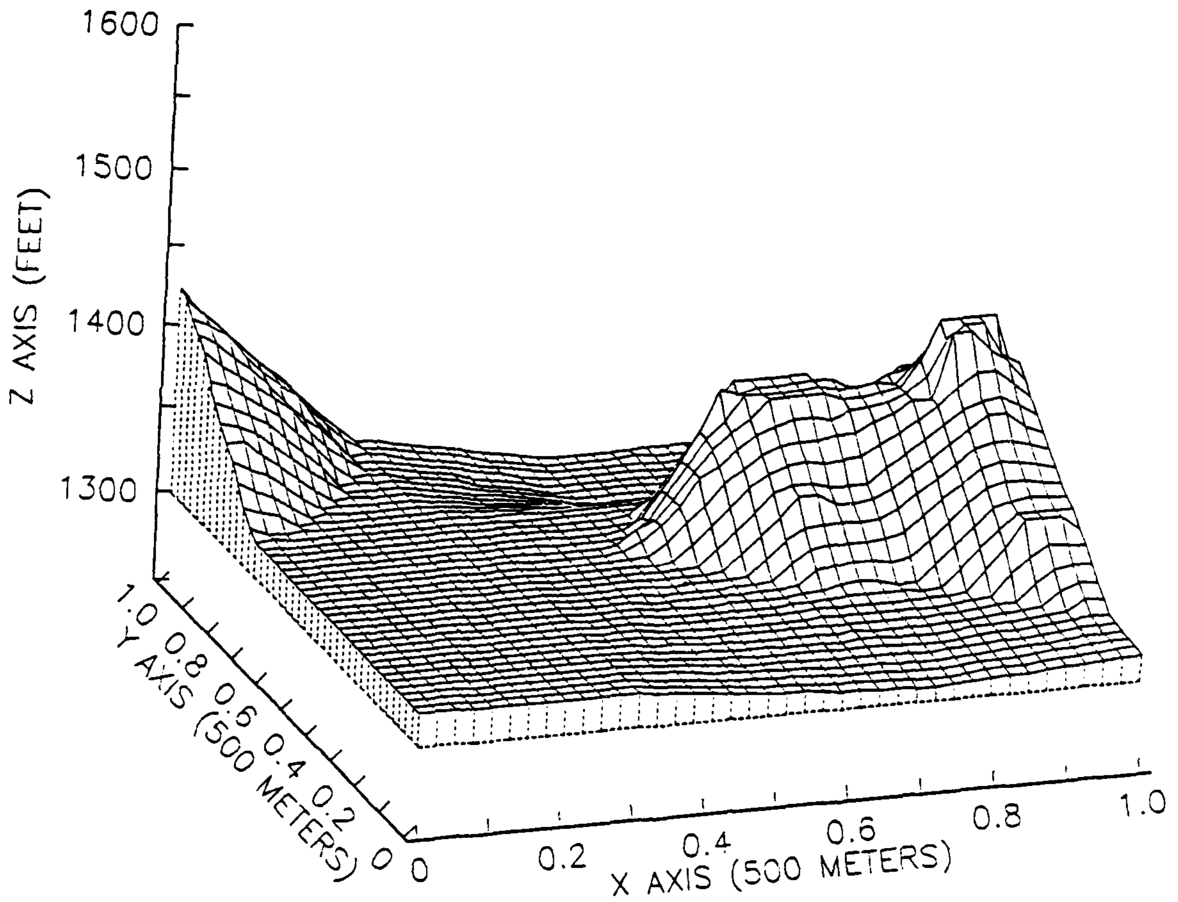


Figure 10. Sample Real Terrain: This is the real terrain that were used in these experiments.

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  1  1  1  1  1  3  3  3  5  5  9  9  9  9  9  9  9  9  9 12 12 12 12 12 12 12 12 12 12 26 26 26 26 26 28 28 28 28  0
  0  1  0  0  0  1  3  0  3  5  5  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 26  0  0  0 26 28  0  0 28  0
  0  1  0  0  0  1  3  3  3  5  5  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 26  0  0  0 26 28  0  0 28  0
  0  1  0  0  0  1  4  4  4  6  6  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 26  0  0  0 26 28  0  0 28  0
  0  1  1  1  1  1  4  4  4  6  6  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 26 26 26 26 26 28 28 28 28  0
  0  2  2  2  2  2  7  7  7  7  7  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 27 27 27 27 27 29 29 29 29  0
  0  2  0  0  0  2  7  0  0  0  7  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  0  0  0  2  7  0  0  0  7  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  0  0  0  2  7  0  0  0  7  9  0  0  0  0  0  0  0  9 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  2  2  2  2  7  7  7  7  7  9  9  9  9  9  9  9  9  9 12 12 12 12 12 12 12 12 12 12 27 27 27 27 27 29 29 29 29  0
  0  8  8  8  8  8  8  8  8  8  8 10 10 10 10 10 10 10 10 10 13 13 13 15 15 21 21 21 21 21 30 30 30 32 32 38 38 38 38  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 13  0 13 15 15 21  0  0  0 21 30  0 30 32 32 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 13 13 13 15 15 21  0  0  0 21 30 30 30 32 32 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 14 14 14 16 16 21  0  0  0 21 31 31 31 33 33 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 14 14 14 16 16 21 21 21 21 21 31 31 31 33 33 38 38 38 38  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 17 17 17 19 19 22 22 22 24 24 34 34 34 36 36 39 39 41 41  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 17 17 17 19 19 22 22 22 24 24 34 34 34 36 36 39 39 41 41  0
  0  8  0  0  0  0  0  0  0  0  8 10  0  0  0  0  0  0  0 10 18 18 18 20 20 23 23 23 25 25 35 35 35 37 37 40 40 42 42  0
  0  8  8  8  8  8  8  8  8  8  8 10 10 10 10 10 10 10 10 10 18 18 18 20 20 23 23 23 25 25 35 35 35 37 37 40 40 42 42  0
  0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 43 43 43 45 45 48 48 48 48 48 51 51 51 51 51 56 56 58 58  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 43  0 43 45 45 48  0  0  0 48 51  0  0  0 51 56 56 58 58  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 43 43 43 45 45 48  0  0  0 48 51  0  0  0 51 56 56 58 58  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 44 44 44 46 46 48  0  C  0 48 51  0  0  0 51 57 57 59 59  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 44 44 44 46 46 48 48 48 48 48 51 51 51 51 51 57 57 59 59  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 47 47 47 47 47 49 49 49 49 49 52 52 52 54 54 60 60 62 62  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 47  0  0  0 47 49  0  0  0 49 52  0 52 54 54 60 60 62 62  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 47  0  0  0 47 49  0  0  0 49 52 52 52 54 54 60 60 62 62  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 47  0  0  0 47 49  0  0  0 49 53 53 53 55 55 61 61 63 63  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 47 47 47 47 47 49 49 49 49 49 53 53 53 55 55 61 61 63 63  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50 50 50 50 50 50 50 50 50 50 64 64 64 64 64 66 66 68 68  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 66 66 68 68  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 66 66 68 68  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 67 67 69 69  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 64 64 64 64 64 67 67 69 69  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 65 65 65 65 65 70 70 70 70  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 65  0  0  0 65 70  0  0 70  0
  0 11  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 11 50  0  0  0  0  0  0  0  0 50 65  0  0  0 65 70  0  0 70  0
  0 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 50 50 50 50 50 50 50 50 50 50 65 65 65 65 65 70 70 70 70  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 11. Subregions Created by the Quadtree Method in the First Approach: Thresholds: low_bound = 0.1; upper_bound = 0.6; standard_deviation = 3. The above picture shows the initial subregions created by quadtree algorithm.

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  1  1  1  1  1  1  1  1  5  5  7  7  7  7  7  7  7  7  7 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12  0
  0  1  0  0  0  0  0  0  1  5  5  7  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0
  0  1  0  0  0  0  0  0  1  5  5  7  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0
  0  1  0  0  0  0  0  0  1  6  6  7  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 12  0
  0  1  1  1  1  1  1  1  1  6  6  7  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0 12 12 12 12 12 12 12 12 12 12  0
  0  2  2  2  2  2  7  7  7  7  7  7  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0 12 27 27 27 27 27 29 29 29 29  0
  0  2  0  0  0  2  7  0  0  0  0  0  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  0  0  0  2  7  0  0  0  0  0  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  0  0  0  2  7  0  0  0  0  0  0  0  0  0  0  0  0  7 12  0  0  0  0  0  0  0  0 12 27  0  0  0 27 29  0  0 29  0
  0  2  2  2  2  2  7  7  7  7  7  7  7  7  7  7  7  7  7  7 12 12 12 12 12 12 12 12 12 12 27 27 27 27 27 29 29 29 29  0
  0  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8 13 13 13 15 15 21 21 21 21 21 30 30 30 32 32 38 38 38 38  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 13  0 13 15 15 21  0  0  0 21 30  0 30 32 32 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 13 13 13 15 15 21  0  0  0 21 30 30 30 32 32 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 14 14 14 16 16 21  0  0  0 21 31 31 31 33 33 38  0  0 38  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 14  0 14 16 16 21 21 21 21 21 31 31 31 33 33 38 38 38 38  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 14  0 14 19 19 22 22 22 22 22 34 34 34 36 36 39 39 41 41  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 14 14 14 19 19 22 22 22 22 22 34 34 34 36 36 39 39 41 41  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 18 18 18 20 20 23 23 23 25 25 35 35 35 37 37 40 40 42 42  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 18 18 18 20 20 23 23 23 25 25 35 35 35 37 37 40 40 42 42  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 43 43 43 45 45 48 48 48 48 48 51 51 51 51 51 56 56 58 58  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 43  0 43 45 45 48  0  0  0 48 51  0  0  0 51 56 56 58 58  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 43 43 43 45 45 48  0  0  0 48 51  0  0  0 51 56 56 58 58  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  8  8  8 46 46 48  0  0  0 48 51  0  0  0 51 57 57 59 59  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 46 46 48 48 48 48 48 51 51 51 51 51 57 57 59 59  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  8  8 49 49 49 49 49 52 52 52 54 54 60 60 62 62  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 49  0  0  0 49 52  0 52 54 54 60 60 62 62  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 49  0  0  0 49 52 52 52 54 54 60 60 62 62  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 49  0  0  0 49 53 53 53 55 55 61 61 63 63  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8  8  8  8  8  8 49 49 49 49 49 53 53 53 55 55 61 61 63 63  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50 50 50 50 50 50 50 50 50 50 64 64 64 64 64 66 66 68 68  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 66 66 68 68  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 66 66 68 68  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0 64 64 64 69 69  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0  0  0 64 69 69  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0  0  0 64 64 64  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0  0  0  0  0 64  0
  0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 50  0  0  0  0  0  0  0  0 50 64  0  0  0  0  0  0  0 64  0
  0  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8 50 50 50 50 50 50 50 50 50 50 64 64 64 64 64 64 64 64 64  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 12. Subregions Created after the Merging in the First Approach: Thresholds: abc_difference = 10; standard_deviation = 3; Some of the initial subregions created by quadtree algorithm are merged by merging process, thus yielding less subregions.

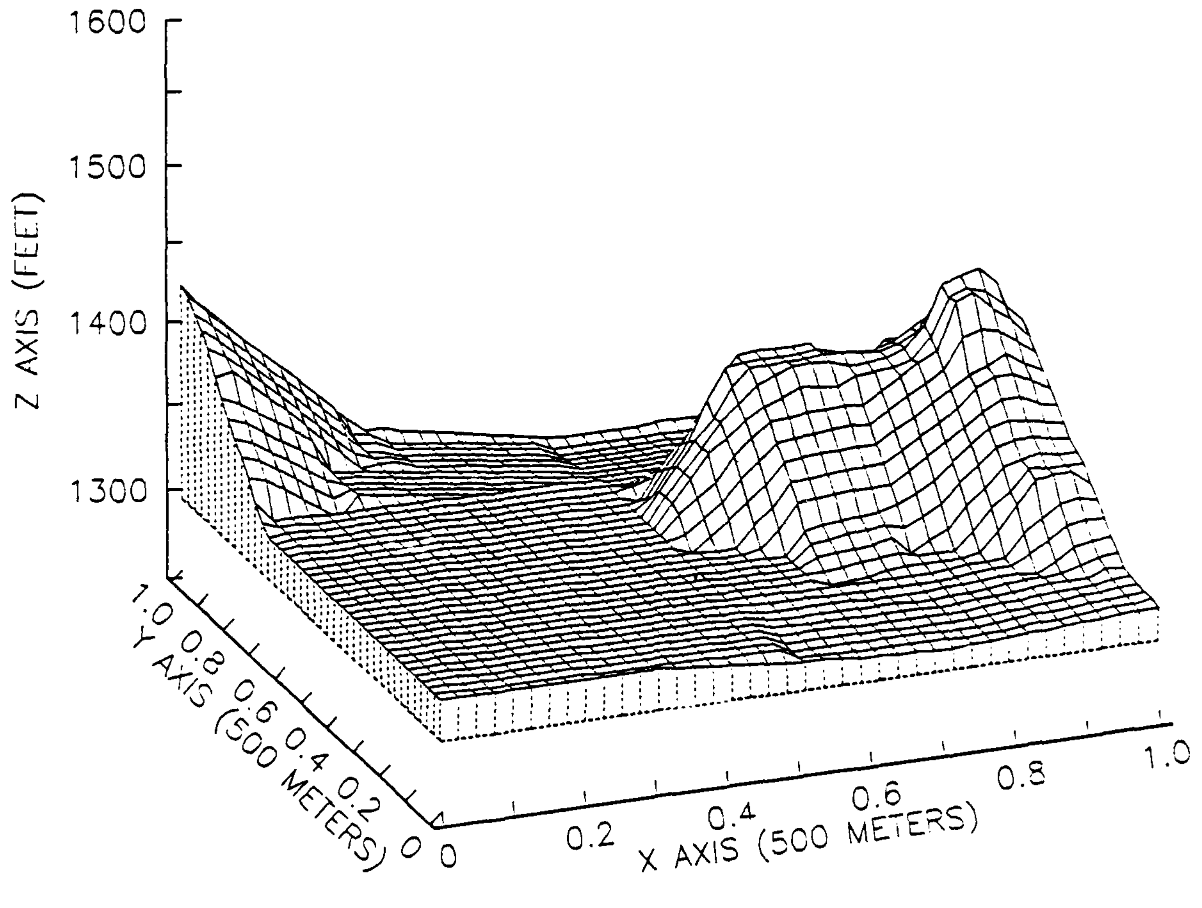


Figure 13. Terrain Model Created by the First Approach: This is the final product of the first approach.

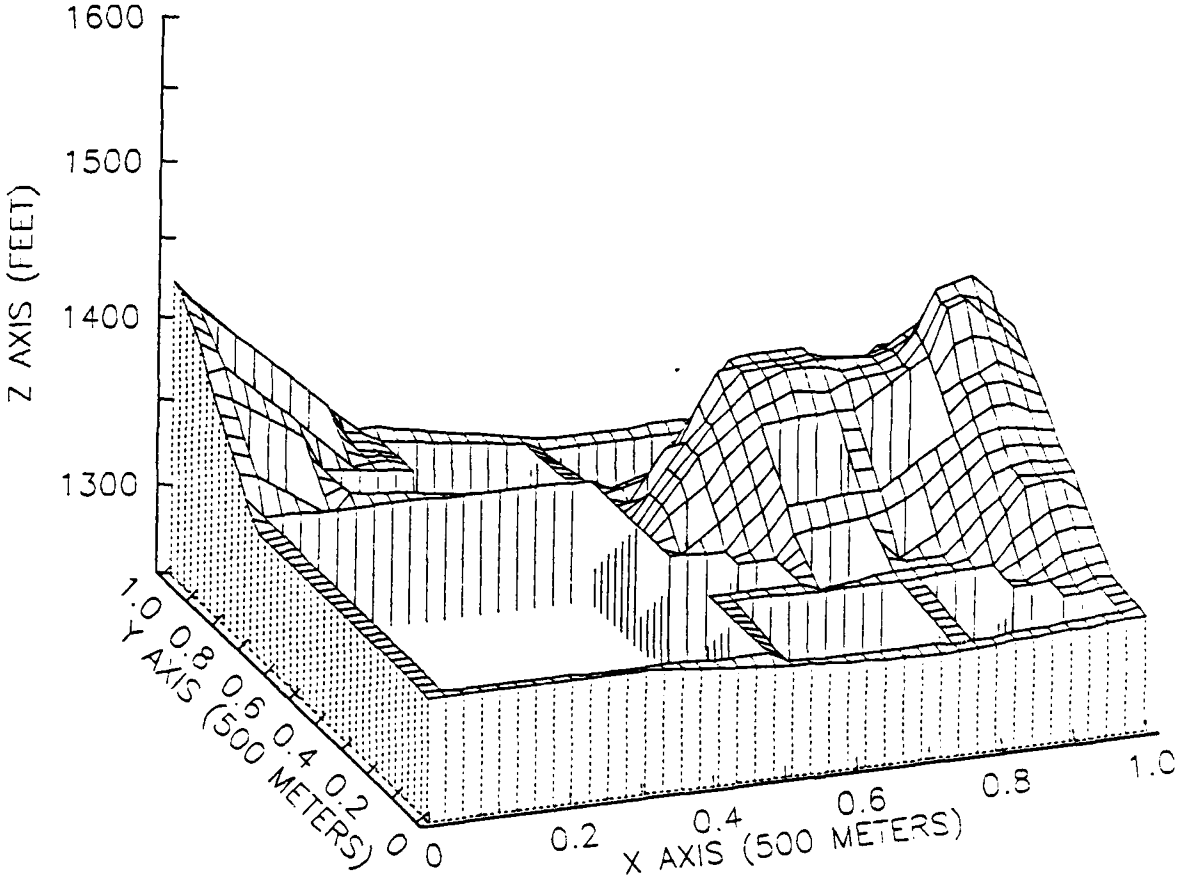


Figure 14. Terrain Model Created by the First Approach Showing Boundaries

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  1  1  1  1  1  3  3  3  3  3  6  6  6  6  6  6  6  6  6  9  9  9  9  9  9  9  9  9  9 17 17 17 17 17 17 17 17 17  0
 0  1  0  0  0  1  3  0  0  0  3  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  1  0  0  0  1  3  0  0  0  3  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  1  0  0  0  1  3  0  0  0  3  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  1  1  1  1  1  3  3  3  3  3  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  2  2  2  2  2  4  4  4  4  4  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  2  0  0  0  2  4  0  0  0  4  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  2  0  0  0  2  4  0  0  0  4  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  2  0  0  0  2  4  0  0  0  4  6  0  0  0  0  0  0  0  6  9  0  0  0  0  0  0  0  0  9 17  0  0  0  0  0  0  0 17  0
 0  2  2  2  2  2  4  4  4  4  4  6  6  6  6  6  6  6  6  6  9  9  9  9  9  9  9  9  9  9 17 17 17 17 17 17 17 17 17  0
 0  5  5  5  5  5  5  5  5  5  5  7  7  7  7  7  7  7  7  7 10 10 10 12 12 15 15 15 15 15 18 18 18 18 18 20 20 20 20  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 10  0 10 12 12 15  0  0  0 15 18  0  0  0 18 20  0  0 20  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 10 10 10 12 12 15  0  0  0 15 18  0  0  0 18 20  0  0 20  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 11 11 11 13 13 15  0  0  0 15 18  0  0  0 18 20  0  0 20  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 11 11 11 13 13 15 15 15 15 15 18 18 18 18 18 20 20 20 20  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 14 14 14 14 14 16 16 16 16 16 19 19 19 19 19 21 21 23 23  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 14  0  0  0 14 16  0  0  0 16 19  0  0  0 19 21 21 23 23  0
 0  5  0  0  0  0  0  0  0  0  5  7  0  0  0  0  0  0  0  7 14  0  0  0 14 16  0  0  0 16 19  0  0  0 19 22 22 24 24  0
 0  5  5  5  5  5  5  5  5  5  5  7  7  7  7  7  7  7  7  7 14 14 14 14 14 16 16 16 16 16 19 19 19 19 19 22 22 24 24  0
 0  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8 25 25 25 25 25 27 27 27 27 27 30 30 30 30 30 30 30 30 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 25  0  0  0 25 27  0  0  0 27 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 25  0  0  0 25 27  0  0  0 27 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 25  0  0  0 25 27  0  0  0 27 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 25 25 25 25 25 27 27 27 27 27 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 26 26 26 26 26 28 28 28 28 28 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 26  0  0  0 26 28  0  0  0 28 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 26  0  0  0 26 28  0  0  0 28 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 26  0  0  0 26 28  0  0  0 28 30  0  0  0  0  0  0  0 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 26 26 26 26 26 28 28 28 28 28 30 30 30 30 30 30 30 30 30  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29 29 29 29 29 29 29 29 29 29 31 31 31 31 31 31 31 31 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  ^  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  8 29  0  0  0  0  0  0  0  0 29 31  0  0  0  0  0  0  0 31  0
 0  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8 29 29 29 29 29 29 29 29 29 29 31 31 31 31 31 31 31 31 31  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 15. Subregions Created by the Quadtree Method in the First Approach: Thresholds: low_bound = 0.1; upper_bound = 0.6; standard_deviation = 7. The above picture shows initial subregions created by the quadtree algorithm using a different threshold (the standard_deviation is changed).

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  1  1  1  1  1  1  1  1  1  1  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  0
  0  1  0  0  0  0  0  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  0  0  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  0  0  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  1  1  1  1  1  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  1  4  4  4  4  4  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  0  0  0  1  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  1  1  1  1  1  4  0  0  0  0  0  0  0  0  0  0  0  0  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  0
  0  4  4  4  4  4  4  0  0  0  0  0  0  0  0  0  0  0  0  4 10 10 10 12 12 15 15 15 15 15 15 15 15 15 15 15 15 15 15  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 10  0 10 12 12 15  0  0  0  0  0  0  0  0  0  0  0  0 15  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 10 10 10 12 12 15  0  0  0  0  0  0  0  0  0  0  0  0 15  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 11 11 11 13 13 15  0  0  0  0  0  0  0  0  0  0  0  0 15  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 11 11 11 13 13 15 15 15 15 15 15 15 15 15 15 15 15 15 15  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 14 14 14 14 14 16 16 16 16 16 19 19 19 19 19 21 21 23 23  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 14  0  0  0 14 16  0  0  0 16 19  0  0  0 19 21 21 23 23  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 14  0  0  0 14 16  0  0  0 16 19  0  0  0 19 22 22 24 24  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 14 14 14 14 14 16 16 16 16 16 19 19 19 19 19 22 22 24 24  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  4  4  4  4  4 27 27 27 27 27 22 22 22 22 22 22 22 22 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 27  0  0  0 27 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 27  0  0  0 27 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 27  0  0  0 27 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 27 27 27 27 27 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 28 28 28 28 28 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 28  0  0  0 28 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 28  0  0  0 28 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 28  0  0  0 28 22  0  0  0  0  0  0  0 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 28 28 28 28 28 22 22 22 22 22 22 22 22 22  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4  0
  0  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 16. Subregions Created after the Merging in the First Approach: Thresholds: abc_difference = 30; standard_deviation = 7; Some of the initial subregions created by quadtree algorithm are merged by the merging process; The thresholds standard_deviation and the abc_difference are changed.

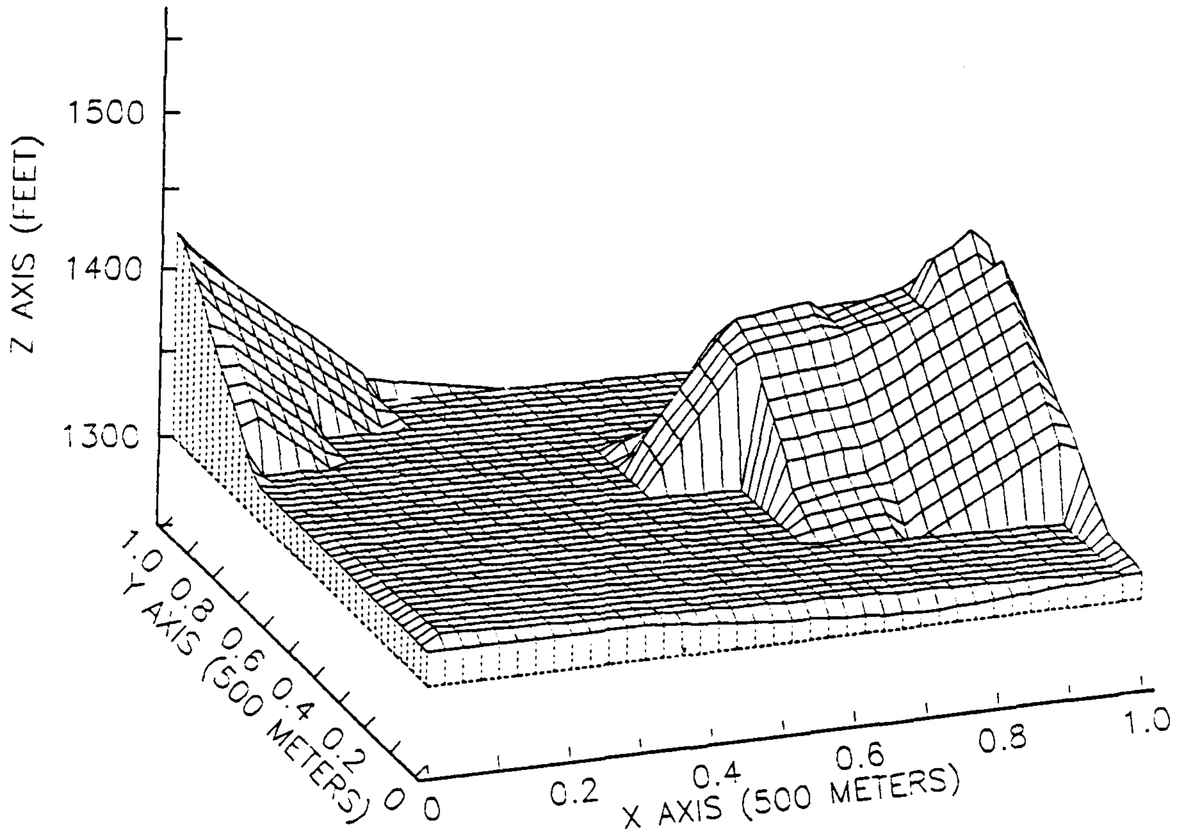


Figure 17. Terrain Model Created by the First Approach: Final product of the first approach using a different set of thresholds.

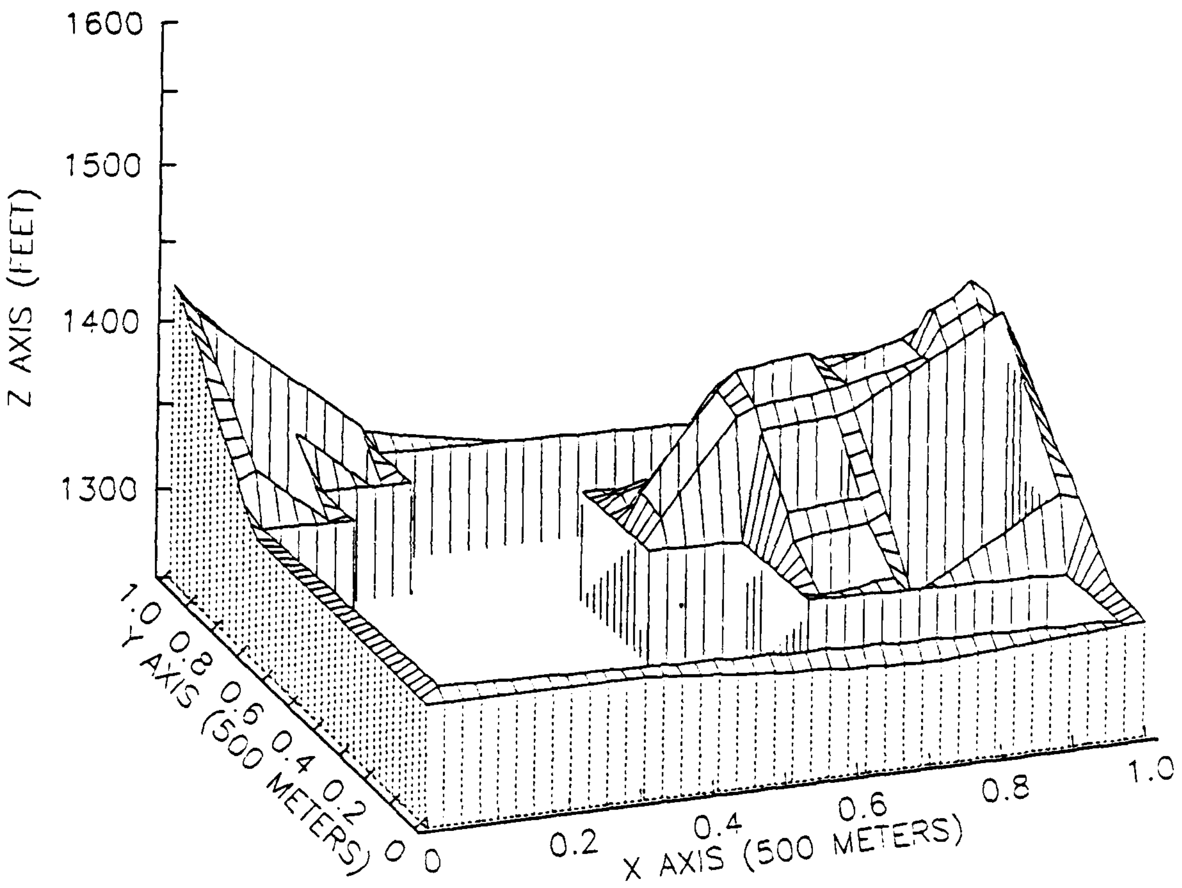


Figure 18. Terrain Model Created by the First Approach Showing Boundaries: The planar patches are different from the planar patches produced from the previous experiment.

3. EXPERIMENTAL RESULTS 1 OF THE SECOND APPROACH

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  1  1  2  3  4  5  6  6  7  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  8  0
0  9  9  9  9  9 10 11 11 12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13  0
0 14 15 15 16 16 17 17 18 19 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20  0
0 21 22 23 23 23 23 24 25 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26  0
0 27 28 28 28 29 30 31 32 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33 33  0
0 34 35 36 36 36 37 38 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39 39  0
0 40 41 42 42 42 43 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 45  0
0 46 47 48 48 49 50 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 52 53 54 54 55  0
0 56 56 57 58 59 60 60 60 60 60 60 60 60 60 60 60 60 60 60 60 61 62 63 63 63 63 64 64 64 65 66 67 67 68 68 69 69 70  0
0 71 72 73 74 75 75 75 75 75 75 75 75 75 75 75 75 75 75 75 75 76 77 78 78 78 78 78 79 80 81 81 82 82 83 84 84 84 85  0
0 86 87 88 89 89 89 89 89 89 89 89 89 89 89 89 89 89 89 89 89 89 90 91 92 93 93 93 93 93 93 93 93 93 94 95 95 96 97  0
0 98 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99 99. 0. 1. 2. 3. 4. 5. 5. 5. 5. 6. 7. 7. 8. 8. 8. 8. 9.10  0
0.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.11.12.13.14.15.15.16.16.16.17.17.18.18.19.20.21.21.22.23  0
0.24.24.24.24.24.24.24.24.24.24.24.24.24.24.24.24.24.24.24.25.26.27.28.29.29.30.30.30.31.32.33.33.34.35.36.37.37.38  0
0.39.39.39.39.39.39.39.39.39.39.39.39.39.39.39.39.39.39.39.40.41.42.43.44.45.46.46.46.47.48.48.48.49.50.51.52.52.53  0
0.54.54.54.54.54.54.54.54.54.54.54.54.54.54.54.54.54.54.54.55.56.56.56.57.58.59.59.59.59.60.61.61.61.62.62.63.63.64  0
0.65.65.65.65.65.65.65.65.65.65.65.65.65.65.65.65.65.65.66.67.68.68.68.69.70.71.71.71.71.71.71.71.72.73.73.74.74.75  0
0.76.76.76.76.76.76.76.76.76.76.76.76.76.76.76.76.76.76.77.78.79.79.80.80.81.82.83.83.84.85.85.85.86.87.88.89.90.91  0
0.92.92.92.92.92.92.92.92.92.92.92.92.92.92.92.92.92.92.93.94.95.96.97.98.99.99: 0: 1: 2: 3: 4: 5: 6: 7: 8: 9:10:11  0
0:12:12:12:12:12:12:12:12:12:12:12:12:12:12:12:12:12:12:13:14:15:15:16:17:18:18:18:19:19:20:20:21:22:23:24:25:26:27  0
0:28:28:28:28:28:28:28:28:28:28:28:28:28:28:28:28:28:28:28:29:30:31:31:32:33:34:34:35:36:36:36:36:37:38:39:40:41:42  0
0:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:44:45:46:46:47:48:49:50:51:52:53:54:54:54:55:56:56:57  0
0:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:58:59:60:61:62:63:64:65:66:67:68:68:68:68:69:70:70:71  0
0:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:72:73:74:75:76:77:78:79:80:81:82:82:82:83:83:83:83  0
0:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:84:85:86:86:87:87:87:88:89:90:90:90:91:92:93:93  0
0:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:95:96:97:97:97:98:99- 0- 0- 0- 1- 2- 3- 4  0
0- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 5- 6- 7- 7- 7- 7- 8- 9-10-10-11-12-13-14  0
0-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-16-17-18-19-20-21-21  0
0-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-22-23-24-25-26-27-27  0
0-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-28-29-30-30-31-32  0
0-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-33-34-35-36-36-37  0
0-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-38-39-40-41  0
0-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-42-43-43  0
0-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44-44  0
0-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45-45  0
0-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46-46  0
0-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47-47  0
0-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48-48  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 19. Subregions Created by the Region Growing Algorithm in the Second Approach: Thresholds: low_bound = 0.1; upper_bound = 0.6; magnitude_ratio = 0.1. This picture is produced after performing the one-dimensional region growing process (row region growing).

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0 22 22  2  2  4  4 22 22- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8  0
 0 22 22 22 22 22  4 22 22- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0 14 22  0  0 22 22 22 22- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0 21 22  0 22 22 22 24 24- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0 21 22 22 22 29 29- 8- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0 34 34 22 22 22 37- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8  0
 0 40 41 22 22 22- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8- 8- 8 45  0
 0 40 47 22 22- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8 52 52 54 54 55  0
 0 47 47 22- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8 61 62 63 63 63 63 64 64 64 65 65 67 67 68 68 54 54 55  0
 0 71 71- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8 61 62 63 63 63 63 63- 8- 8 81 81 67 67 68 84 84 84 84  0
 0- 8- 8- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8 62- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8 95 95 96 97  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8. 3. 4. 5. 5. 5. 5. 6. 7. 7. 8. 8. 8. 8 96 96  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8.13.13. 4. 4. 5  0. 5. 6. 6.18.18.18.18. 8. 8 96 96  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8.27.28.29.29. 5. 5. 5. 6.32.33.33.18.18.36 96 96 96  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8.27.27.43.44.45.45.45.45.45.33.33.33.36.36.36.36.36.36  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8.55.27.27.27.45.45.59.59.59.59.60.61.61.61.62.62.63.63.64  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8.55.27.27.27.69.45.59.59.59.59.59.59.59.61.73.73.63.63.75  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8.77.55.79.79.80.80.81.82.82.82.82.59.59.59.61.87.88.88.88.88  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8.77.94.79.96.80.98.99.99.82.82.82.82.82.59.61.87.88.88.88:11  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8.94.79.79:16.98:18:18:18:19:19:20:20:37:37:37.88.88:26:27  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8:16:16:35:35:18:18:35:36:36:36:36:37:37:39:39:39:27  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8:16:16:35:35:35:35:35:35:36:54:54:54:39:56:56:57  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8:16:62:35:35:35:35:67:54:54  0:54:69:69:69:71  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8:62:62:62:62:79:67:81:54  0:54:71:71:71:71  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8:86:86:79:79:79:67:81:54  0:54:71:92:93:93  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8:79:79:79:67:81:54:54:54:71:92- 3- 3  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 7- 7- 7- 7- 8- 8:54:54:71-12- 3- 3  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8- 8- 8- 8- 8- 8:54-19-12-21-21  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8-24-25-21-21-21  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8-25-25-31-31  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8-36-36-37  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8-41  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8-43-43  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8- 8- 8  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0- 8  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0- 8  0
 0- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 20. **Subregions Created after the Merging in the Second Approach:** Threshold: abc_difference = 10; the two-dimensional region growing algorithm is applied over the subregions created by the one-dimensional region growing algorithm and then merging is performed.

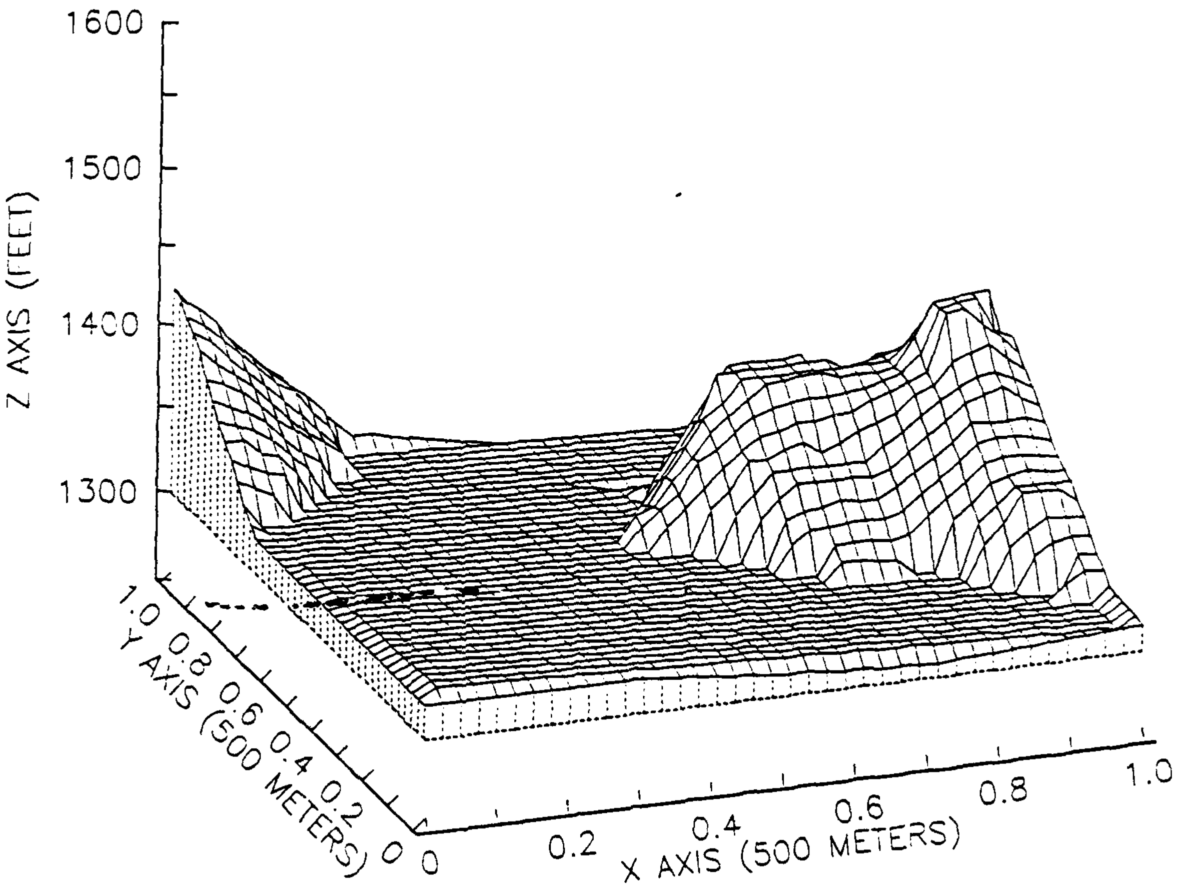


Figure 21. Terrain Model Created by the Second Approach: Final product of the second approach.

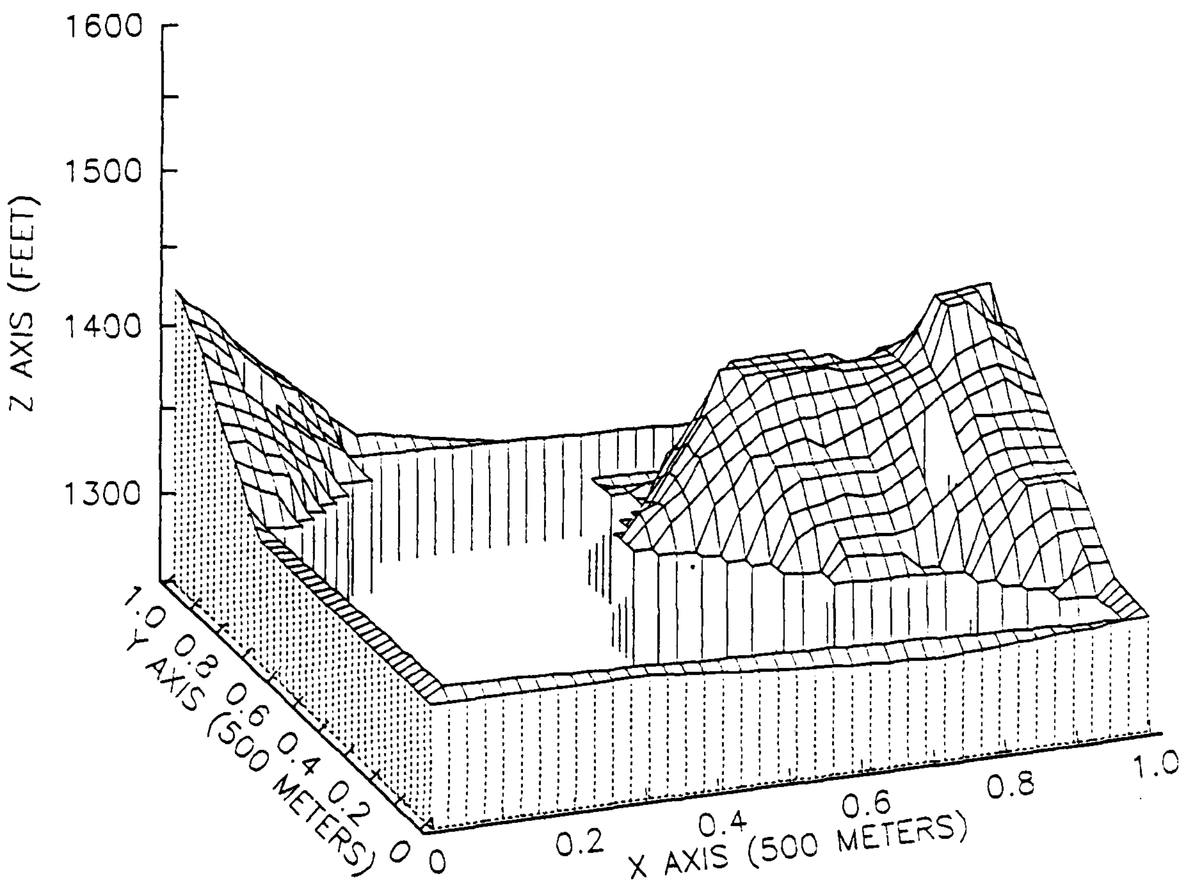


Figure 22. Terrain Model Created by the Second Approach Showing Boundaries

4. EXPERIMENTAL RESULTS 2 OF THE SECOND APPROACH

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 2 2 3 4 4 4 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 0
0 7 7 7 7 7 8 9 9 10 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 11 0
0 12 13 13 13 13 13 13 13 14 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 0
0 16 17 18 18 18 18 19 20 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 0
0 22 23 23 23 24 25 26 27 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 28 0
0 29 30 31 31 31 32 33 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 34 0
0 35 36 36 36 36 37 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 38 39 0
0 40 40 41 41 42 43 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 44 45 45 46 46 46 0
0 47 47 48 49 50 51 51 51 51 51 51 51 51 51 51 51 51 51 51 51 52 53 54 54 54 54 55 55 55 56 57 58 58 59 59 60 60 60 0
0 61 62 62 63 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 64 65 66 67 67 67 67 67 68 69 70 70 71 71 72 72 72 72 72 0
0 73 74 75 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 77 78 79 79 79 79 79 79 79 79 79 79 80 81 81 82 83 0
0 84 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 85 86 87 88 89 89 90 90 90 90 91 92 92 93 93 93 93 94 95 0
0 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 96 97 98 99 99 99. 0. 0. 0. 1. 1. 1. 1. 2. 3. 3. 3. 4. 5 0
0. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 7. 8. 9. 9.10.10.11.11.11.12.12.13.13.14.15.16.17.17.18 0
0.19.19.19.19.19.19.19.19.19.19.19.19.19.19.19.19.19.19.19.20.21.21.22.23.24.25.25.25.26.27.27.27.28.28.29.30.30.31 0
0.32.32.32.32.32.32.32.32.32.32.32.32.32.32.32.32.32.32.32.33.34.34.34.35.36.37.37.37.37.38.38.38.38.39.39.40.40.41 0
0.42.42.42.42.42.42.42.42.42.42.42.42.42.42.42.42.42.42.43.44.45.45.45.46.47.48.48.48.48.48.48.48.49.50.50.51.51.52 0
0.53.53.53.53.53.53.53.53.53.53.53.53.53.53.53.53.53.53.54.55.56.56.57.57.58.59.60.60.60.61.61.61.62.63.64.65.66.67 0
0.68.68.68.68.68.68.68.68.68.68.68.68.68.68.68.68.68.68.69.70.71.72.73.74.75.75.76.77.78.79.80.81.82.83.84.85.86.87 0
0.88.88.88.88.88.88.88.88.88.88.88.88.88.88.88.88.88.88.89.90.91.91.92.92.93.93.93.94.94.95.95.95.96.97.98.99.99: 0 0
0: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 1: 2: 3: 4: 4: 5: 5: 6: 6: 6: 7: 7: 7: 7: 7: 8: 8: 8: 8: 9 0
0:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:10:11:12:13:13:14:15:15:16:17:18:19:20:20:20:21:21:21:21 0
0:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:22:23:24:25:26:27:27:28:29:30:31:31:31:31:32:32:32:33 0
0:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:34:35:36:36:36:36:37:38:39:40:41:41:41:42:42:42:42 0
0:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:43:44:45:45:46:46:46:47:48:49:49:49:50:50:51:51 0
0:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:52:53:54:55:55:55:56:57:58:58:58:59:60:60:61 0
0:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:62:63:63:63:63:63:64:65:66:66:67:68:69:70 0
0:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:71:72:73:74:75:76:77:77 0
0:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:78:79:80:80:81:82:82 0
0:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:83:84:85:85:85:86 0
0:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:87:88:89:90:90:91 0
0:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:92:93:94:94 0
0:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:95:96:96 0
0:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97:97 0
0:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98:98 0
0:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99:99 0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0 0
0- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1- 1 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Figure 23. Subregions Created by the Region Growing Algorithm in the Second Approach: Thresholds: low_bound = 0.1; upper_bound = 0.6; magnitude_ratio = 0.2. This picture is produced after performing one-dimensional region growing using a different threshold (the magnitude_ratio is changed).

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0 16 16 16 16  3 16 16 16:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80  0
0 16 16  0 16 16  8 16 16:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0 12 16  0  0 16 16 16 16:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0 16 16  0  0 16 16:80:80:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0 16 16 16  0 16:80:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0 29 29 16  0 16:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80  0
0 35 16 16 16 16:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:80:80 39  0
0 35 35 16 16:80:80  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:80:80:80:80:80  0:80:80:80:80:80 45 45 58 58 58  0
0 35 35 16:80:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 52 52 54 54 54 54:80:80:80 56 56 58 58 58 58 58 58 58  0
0:80 35 35:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 52 52 54 54 54 54 54 68 68 70 70 58 58 72 72 72 72 72  0
0:80:80:80:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80 52 68 68 68 68 68 68 68 68 68 68 68 68 81 81 82 82  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80 89 89 90 90 90 90 90 92 92 81 81 81 81 82 82  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80 89 89 89 90  0  0  0 90 90 90. 2 81 81 81 82 82  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80 89 89.10.10 90 90 90 90 90.13.13.14.15.16 82 82 82  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89 89.22.22.24.24.24.24.24.13.13.13.14.14.16.16.16.16  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89 89 89.35.24.37.37.37.37.38.38.38.38.39.39.40.40.41  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89 89 89.35.24.37.37.37.37.37.37.37.38.50.50.40.40.52  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89 89.57.57.24.24.24.24.24.37.37.37.38.50.64.64.64.64  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89.72.57.74.75.75.24.24.24.24.24.37.38.83.64.64.64.87  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80 89 89:80:80.75.75.75.24.24.95.95.95.96.97.64.99.99.87  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:80.75.75.75.75.75: 7: 7: 7: 7: 7.97.97.97.97.87  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80.75:15:15:15: 7:18: 7:20:20:20:21:21:21:21  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:15:15:28:28:30:20:20  0:20:32:32:32:21  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:15:15:15:15:28:28:30:40:20  0:20:21:21:21:21  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:15:15:28:28:28:30:40:20  0:20:21:21:51:51  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:28:28:28:30:40:20:20:20:21:60:60:61  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:80:80:80:80:20:20:21:60:69:69  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:20:21:60:69:69  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:69:69:69  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80:86  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:90:90:86  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:86:86  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:96:96  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80:80:80  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0:80  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0:80  0
0:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80:80  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 24. Subregions Created after the Merging in the Second Approach: Threshold values used: abc_difference = 20. Two dimensional region growing is applied and then merging is performed using different thresholds (abc_difference and magnitude_ration are changed).

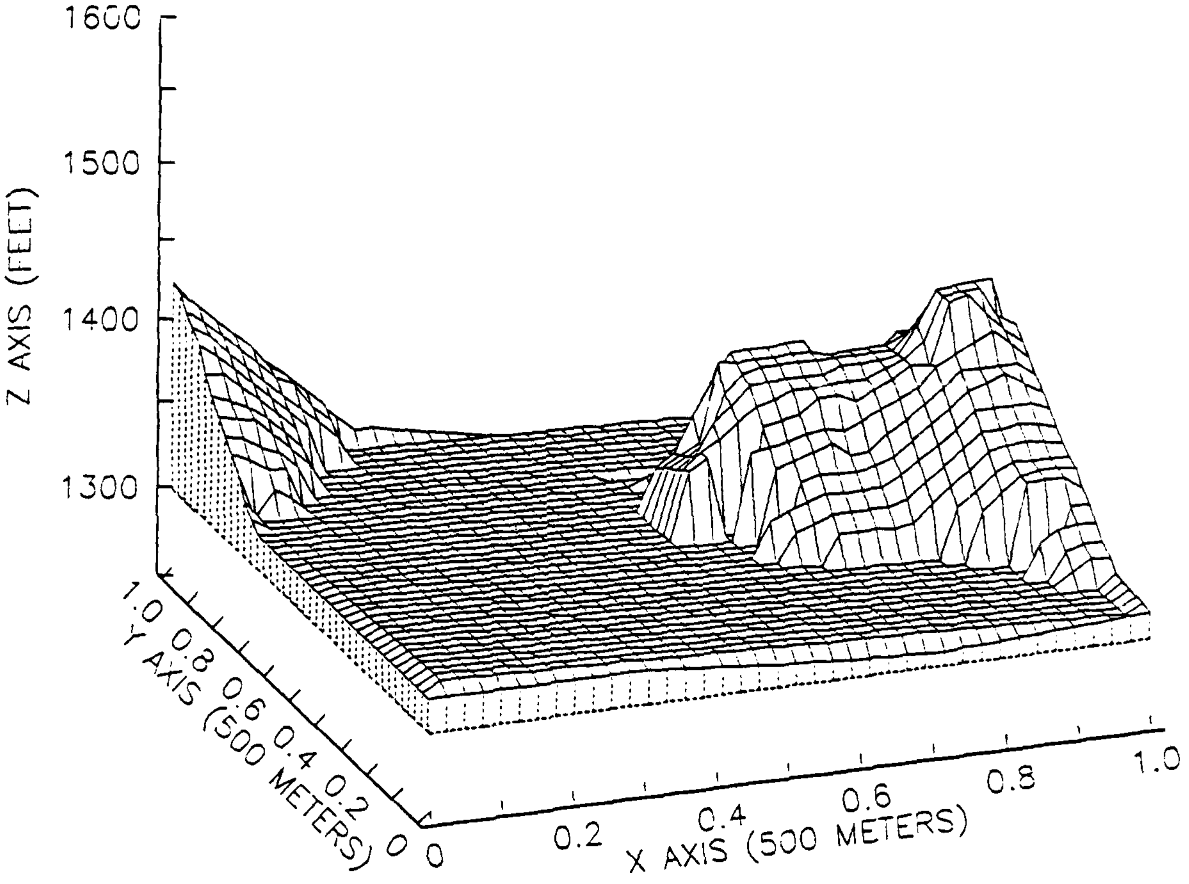


Figure 25. Terrain Model Created by the Second Approach: Final product of the second approach. Notice that this picture is different from the picture produced by the previous experiment.

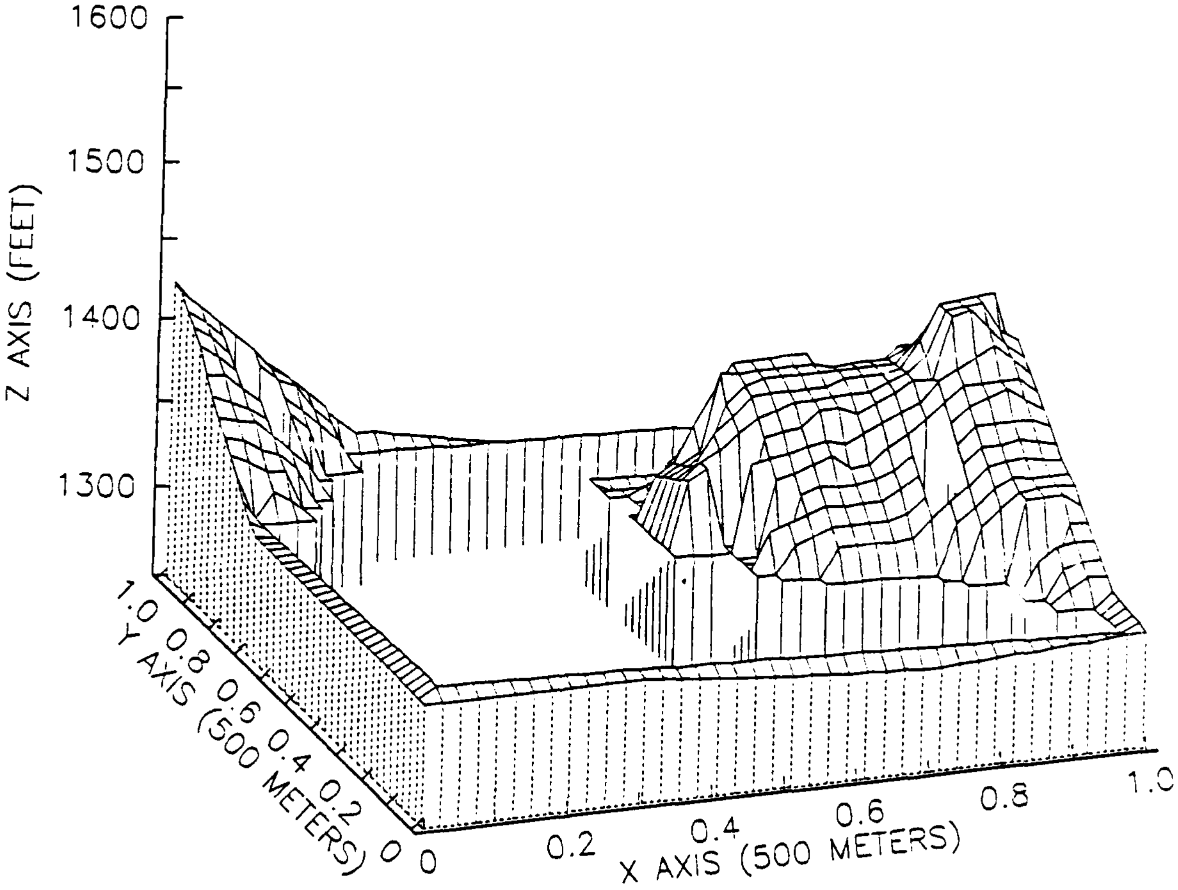


Figure 26. Terrain Model Created by the Second Approach Showing Boundaries

5. EXPERIMENTAL RESULTS OF THE THIRD APPROACH

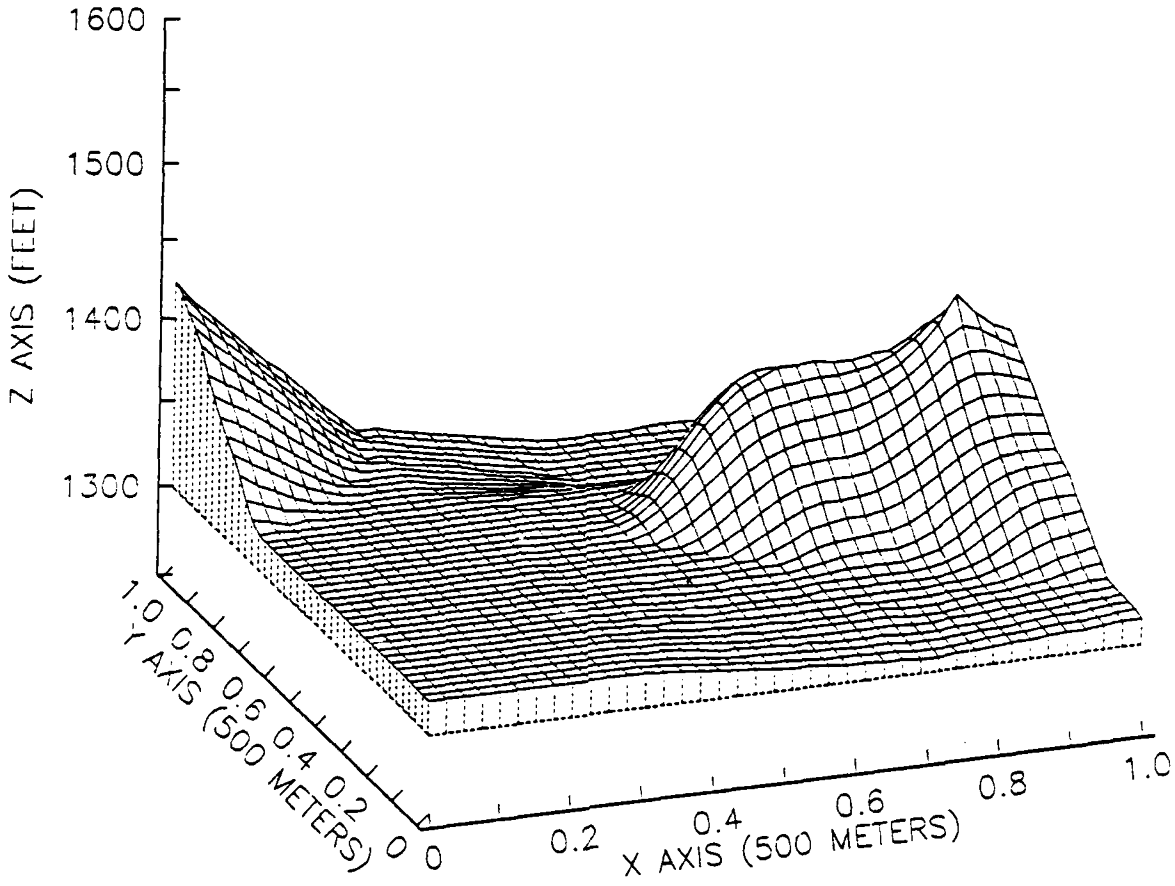


Figure 27. Smoothed Sample Terrain: The original sample terrain is smoothed. Bottom-up modeling will be performed on this smoothed terrain.

```
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  1  2  2  2  2  2  2  2  3  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  4  0
0  5  6  6  6  6  6  6  7  8  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  9  0
0 10 10 10 10 10 10 10 11 12 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13  0
0 14 14 15 15 15 15 16 17 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18  0
0 19 20 20 20 20 20 21 22 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23 23  0
0 24 24 24 24 24 25 26 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27 27  0
0 28 28 28 28 29 30 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 31 32  0
0 33 33 33 34 35 36 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 37 38 39 40 40 40  0
0 41 41 42 43 44 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 46 47 48 48 48 49 49 49 49 49 50 50 50 51 51 51 52 53  0
0 54 55 56 57 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 58 59 60 61 62 63 63 63 63 63 63 63 63 63 64 64 64 64 65  0
0 66 67 68 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 69 70 71 72 73 74 74 74 74 74 74 76 74 74 74 74 74 74 74  0
0 75 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 76 77 78 79 80 81 82 83 84 84 85 85 86 87 88 88 89 90 91  0
0 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 92 93 94 95 95 95 96 97 97 97 97 97 97 97 97 97 97 97 97  0
0 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 98 99. 0. 1. 2. 2. 2. 3. 4. 4. 4. 4. 4. 4. 4. 5. 5. 5. 5. 5  0
0. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 6. 7. 8. 9. 9.10.11.11.12.12.12.12.12.12.13.14.14.15.16.17  0
0.18.18.18.18.18.18.18.18.18 18.18.18.18.18.18.18.18.18.19.20.21.22.23.24.25.26.26.27.27.28.28.29.30.31.31.32.33.34  0
0.35.35.35.35.35.35.35.35.35.35.35.35.35.35.35.35.35.35.36.37.38.39.39.40.41.42.43.43.43.43.43.43.44.45.45.46.47.47  0
0.48.48.48.48.48.48.48.48.48.48.48.48.48.48.48.48.48.48.49.50.51.52.52.53.54.55.55.56.57.58.59.60.61.62.63.64.65.65  0
0.66.66.66.66.66.66.66.66.66.66.66.66.66.66.66.66.66.66.67.68.69.70.70.70.71.72.73.73.73.74.75.76.77.78.78.79.80.81  0
0.82.82.82.82.82.82.82.82.82.82.82.82.82.82.82.82.82.82.83.84.85 56.86.86.87.88.89.89.89.90.91.92.93.94.95.96.97.98  0
0.99.99.99.99.99.99.99.99.99.99.99.99.99.99.99.99.99.99.99: 0: 1: 2: 2: 2: 3: 4: 5: 5: 5: 6: 7: 7: 7: 8: 9:10:10:10  0
0:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:11:12:13:14:14:15:16:17:17:17:18:18:19:19:19:20:21:21:22  0
0:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:23:24:25:26:27:28:29:29:29:30:31:31:31:31:32:33:33:34  0
0:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:35:36:37:38:39:40:40:40:41:42:42:42:42:43:44:44:45  0
0:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:46:47:48:48:49:49:49:50:51:52:52:52:53:54:55:55  0
0:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:56:57:58:58:59:59:60:61:62:62:62:63:64:65:66  0
0:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:67:68:69:70:71:72:73:74:74:75:76:77:78  0
0:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:79:80:81:82:83:83:84:85:85  0
0:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:86:87:88:89:90:91:92:93  0
0:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:94:95:96:97:98:98:99  0
0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 0- 1- 2- 3- 3- 3  0
0- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 4- 5- 6- 6- 7  0
0- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 8- 9-10  0
0-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11-11  0
0-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12-12  0
0-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13-13  0
0-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-1⸍-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14-14  0
0-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15-15  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 28. **Subregions Created by the Third Approach:** Thresholds: low_bound = 0.1; upper_bound = 0.6; magnitude_ratio = 0.1.

```
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  1  2  2  2  2  2  2  2 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62  0
  0  1  2  0  0  0  0  2 62 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0  2  2  2  0  0  2  2 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 14 14  2  0  0  2 62 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 14  2  2  0  0  2 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0  2  2  0  2  2  2 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62  0
  0  2  0  2  2 62 62 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62 62 62 32  0
  0  2  2  2 34 34 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62 62 38 38 40 40 40  0
  0  2  2 34 34 34 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0 46 47 48 48 48  0  0  0  0 62 50 50 50 51 51 51 52 52  0
  0  2 62 62 62 62 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0 46 47 48 62 62 62 62 62 62 62 62 62 62 51 51 51 51 65  0
  0 62 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 46  0  0 62 74 74 74 74 74 74 74 74 74 74 74 74 74 74  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62 81 82 82 90 90 85 85 85 85 88 88 89 90 91  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 81 81 81 96 90 90 90 90 90 90 90 90 90 90 90 90  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 81 81 81 81. 3. 4. 4. 4. 4. 4. 4. 4 90 90 90 90 90  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62. 8 81 81.10.11.11.12.12.12.12.12.12.13.14.14.15.15.15  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.20. 8 81.23.10.25.26.26.12  0  0.12.13.13.31.31.32.32.32  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.20.38.39.39.40.25.42.12.12.12.12.12.12.44.63.63.46.47.47  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.20.51.39.39.40.25.55.55.56.57.57.57.12.44.63.63.46.47.47  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.68.51.39.39.39.71.72.56.56.56.57.57.57.77.63.63.79.80.47  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.68.51.86.86.86.71.72.89.89.89.90.90.90.77.77.95.96.80.80  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62.86.86.86.71.72.89  0.89, 6, 8, 8, 8, 8, 9.96.96.96  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62.86.86.71.72.89  0.89, 6, 6, 8  0: 8, 9.96.96,22  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62.86.71.72.89  0.89,30, 8, 8  0, 8, 9.96.96,34  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62:38.72.89  0.89,30, 8, 8  0, 8,43,44,44,34  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62.72.72.89.89.89,30,51, 8  0, 8,43,44:34,34  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62:58,58,59,59,60,51, 8, 8, 8,43,44:44,66  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62 62 62 62 62 62, 8, 8,43,44,44,78  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62: 8, 8,44,85,85  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62: 8,91:92:93  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62:91,91:93  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62:93,93,93  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62- 6- 6- 7  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62 62 62 62  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 62  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 62  0
  0 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62 62  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Figure 29. **Subregions Created after the Merging in the Third Approach:** Threshold: abc_difference = 10.

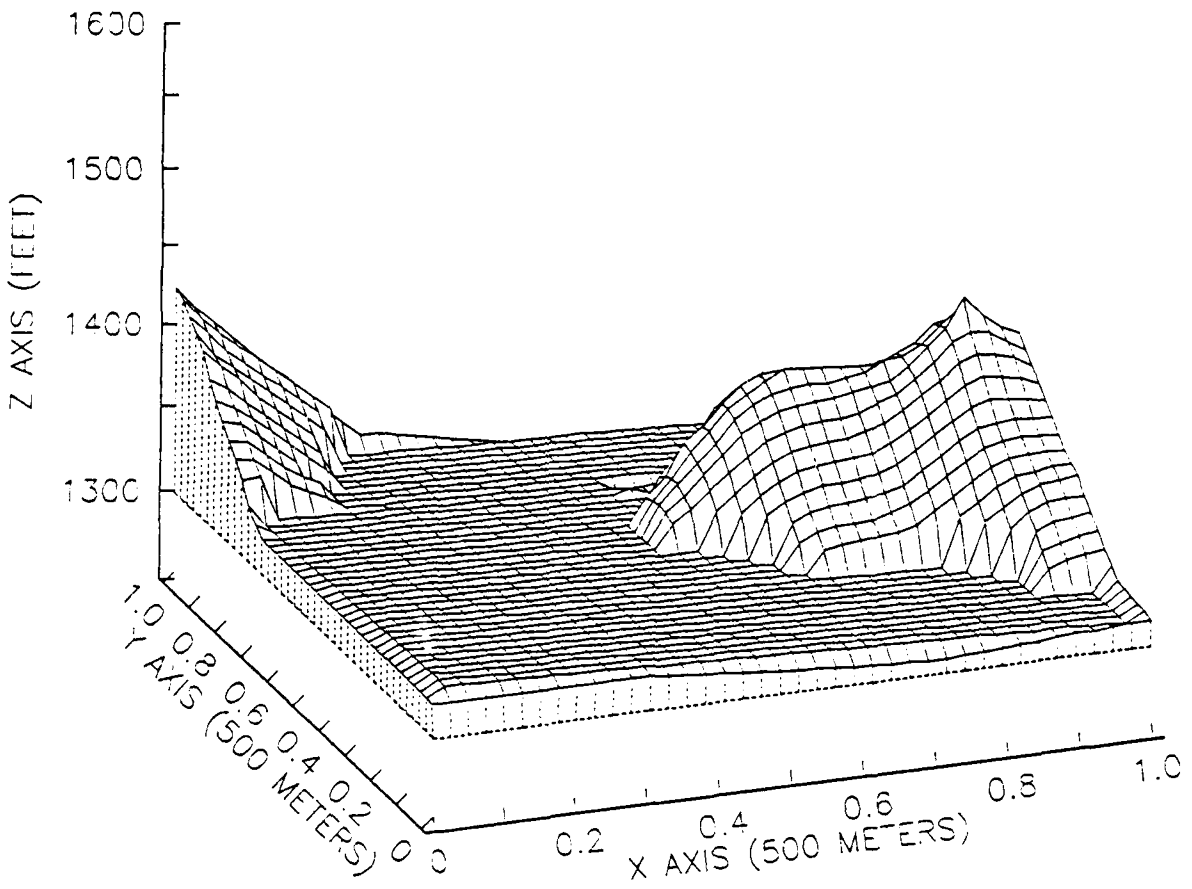


Figure 30. Terrain Model Created by the Third Approach: This picture looks almost the same as the the picture produced by the second approach; this approach produced less subregions than the second approach.

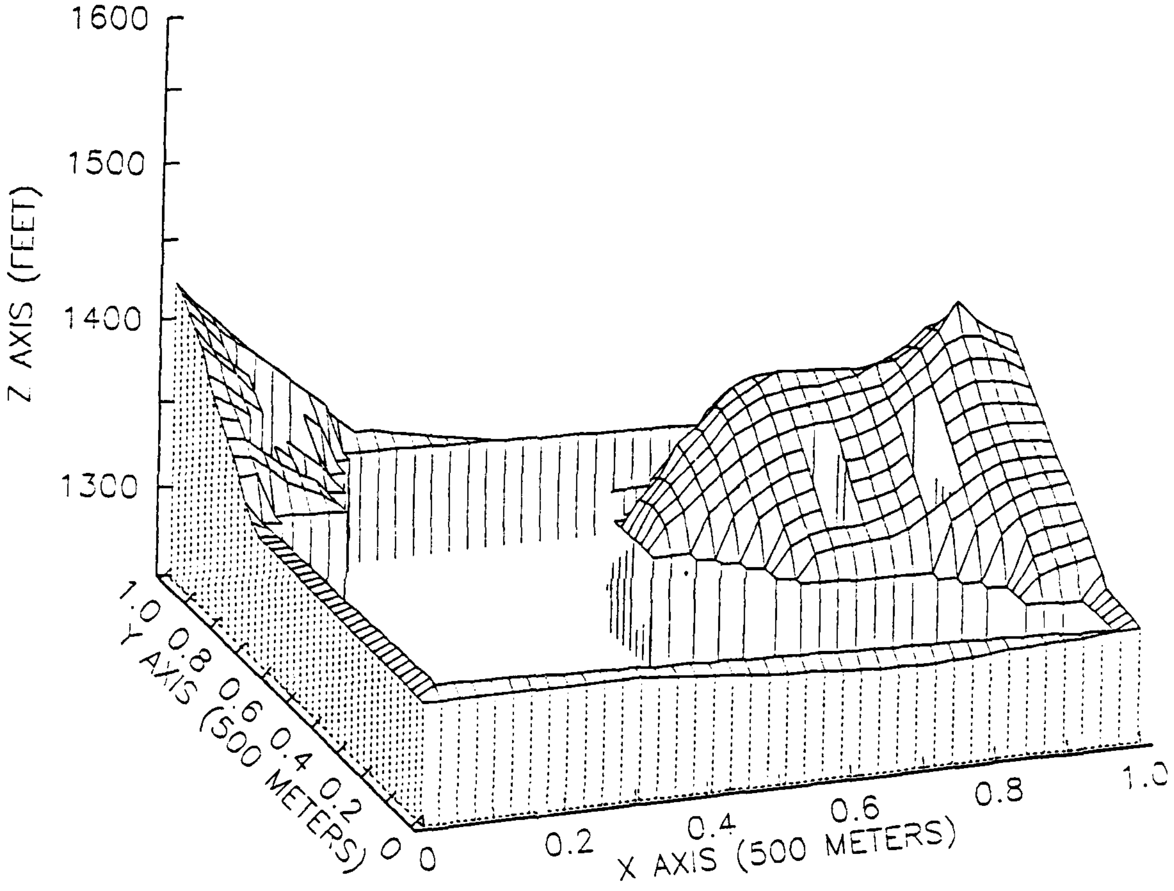


Figure 31. Terrain Model Created by the Third Approach Showing Boundaries

V. CONCLUSIONS

In this research, we explored three types of planar-patch terrain modeling. The purpose of such models is a simple and yet accurate view of terrain. Since our models were developed for route planning, we required a minimal number of subregions, since this directly affects the cost of the route planning.

The first algorithm (joint top-down and bottom-up algorithm) takes the longest execution time among the three. The second (strict bottom-up) and the third (bottom-up algorithm applied to the smoothed elevation data) algorithms take almost same time, though the third algorithm requires the input elevation data to be smoothed first, requiring some extra preprocessing time. As far as result accuracy (simplicity) is concerned, the first algorithm is better than the others. The second and third do not use a standard deviation threshold, resulting in possible inaccuracy, and tend to produce more subregions. Since all of the three algorithms use almost the same data structure, a two-dimensional **point_array** and a one-dimensional **subregion_array**, they take almost the same space. As for the application environments of these algorithms, the first algorithm could be used in situations where the accuracy is critical, while the second and third algoritms could be preferred in situations where a short execution time is required and linear terrain features should be identified.

Since the algorithms used in this research operate on large arrays and involve heavy computations, a multiple-processor computer architecture would yield superior processing capability since the same operations could be performed for each part of the input matrix simumtaneously. As I conclude this research, I regret that I could not solve the continuity issue completely.

APPENDIX A. PASCAL CODE OF JOINT TOP-DOWN AND BOTTOM-UP ALGORITHM

```
(*$s350000*)
(**************************************************************************
   Author   = Yee, Seung Hee.
   Date     = 3 June 1988.
   Input    = 1. Elevation data acquired from Fort. Hunter Liggett
                 data base (40 by 40 square).
              2. file of magnitude of gradient of the above elevation
                 data created by using the program in Appendix 3.
   Output   = 1. Elevation data of the model created by this program.
              2. Elevation data of the boundary points, other points
                 are given the value one so that when we display
                 this model in three-dimensional pictures, we can
                 only see the boundaries of the patches.
   Computer = Waterloo Pascal on IBM 370/3033AP,
              Naval Postgraduate School.
   Description =
              This program is the Pascal implementation of the
              Joint top-down and bottom-up terrain modeling.
**************************************************************************)


program top_down(input,output);
const
     mtinft  = 3.2808;
     low_bound = 0.1;   (* mag > 10 % slope  then divide *)
                       (* mag <= 10 % then disregard *)
     upper_bound = 0.6;   (* mag < 60 % then do not subdivide *)
     max_sd   = 3;
     abc_difference =10 ;
     max_group_num= 300;
     row = 40;
     col = 40;
     increment = 3;
type
   elrange = 1..10000;
   pointrec = record
                el : elrange ;
                mag: real;
                gp : 0..max_group_num;
              end;

   magrec = record
                 mag : real;
                 r,c : 1..row;
              end;

   point_array=array (.1..row,1..col.) of pointrec;

   grouppointer = ¬grouprec ;
   grouprec = record
                 rl,rh,
                 cl,cr : 1..row;
                 magmax,magmin : magrec ;
                 up    : grouppointer;
              end;
   constarray = array (.1..3,1..4.) of real;

   nodepointer = ¬node;
   node  =  record
              r,c :integer;
              next : nodepointer;
           end;

   coefrec = record
                crl,crh,
                ccl,ccr : 1..row;
                a,b,c : real ;
                minmag,maxmag  : real;
                sd   : real ;
                ee,ex,ey : real;
                side : 0..max_group_num;
                constarr : constarray;
                active : boolean;
                list  : nodepointer ;
             end;

   subregion_array = array (.1..max_group_num.) of coefrec ;


var
   points : point_array ;
   groups : grouprec ;
   subregion  : subregion_array ;
   ok     : constarray;
   data,datam,dout1,dout2  : text;
   counter,r,c,i : integer;
   avg,ee,ex,ey  : real;
   top,p : grouppointer;
   temparr: array (.1..max_group_num.) of boolean;

function variance (a,b,c,ee,ex,ey:real; ok:constarray):real;
begin
        variance := (ee-(2*a*ex)-(2*b*ey)-(2*c*ok(.3,4.))
                        +(a*a*ok(.1,1.))+(2*a*b*ok(.2,1.))
                        +(2*b*c*ok(.2,3.))+(2*a*c*ok(.1,3.))
                        +(b*b*ok(.2,2.))+(c*c*ok(.3,3.)))
                        /(ok(.3,3.)-1);
end;

procedure initialize;
var
   r,c :integer;
begin
     counter := 0 ;
     for c := 1 to col do
     for r := 1 to row do
          points(.r.)(.c.).gp := 0 ;
     for r := 1 to max_group_num do
     begin
          subregion(.r.).a := -999;
          subregion(.r.).b := -999;
          subregion(.r.).c := -999;
          subregion(.r.).active := false;
          subregion(.r.).side := 0;
          subregion(.r.).sd := -999;
     end;
end;              (* procedure initialize *)


procedure  getconst(var points:point_array; rl,rh,cl,cr:integer;
                    var arr:constarray; var ee,ex,ey :real);
var
   i,j:integer;

begin
  for i := 1 to 3 do
      begin
         for j := 1 to 4 do
          arr(.i,j.):=0;
      end;
   ee := 0;
   ex := 0;
   ey := 0;
   for i := rl to rh do
   begin
         for j := cl to cr do
         begin
             arr(.1,1.):=arr(.1,1.)+(i*i);
             arr(.1,2.):=arr(.1,2.)+(j*i);
             arr(.1,3.):=arr(.1,3.)+ i;
             arr(.1,4.):=arr(.1,4.)+(points(.i,j.).el*i);

             arr(.2,1.):=arr(.2,1.)+(i*j);
             arr(.2,2.):=arr(.2,2.)+(j*j);
             arr(.2,3.):=arr(.2,3.)+ j;
             arr(.2,4.):=arr(.2,4.)+(points(.i,j.).el*j);

             arr(.3,1.):=arr(.3,1.)+i;
             arr(.3,2.):=arr(.3,2.)+j;
             arr(.3,3.):=arr(.3,3.)+1;
             arr(.3,4.):=arr(.3,4.)+points(.i,j.).el;

             ee := ee + sqr(points(.i,j.).el);
             ex := ex + (points(.i,j.).el * i );
             ey := ey + (points(.i,j.).el * j );

         end  (* j *)
   end;      (* i *)
end;         (* procedure get const *)
```

```
  procedure gauss(ok: constarray; var aa,bb,cc: real);

  var
     i,j : integer;
     temp : array (.1..4.) of real;
     t : real;                          (* error checking purpose *)
  begin

     i := 1;
     while ok(.i,1.) = 0.0 do  i := i + 1 ; (* row rotation *)
     for j := 1 to 4 do
     begin
        temp(.j.) := ok(.1,j.);
        ok(.1,j.) := ok(.i,j.);
        ok(.i,j.) := temp(.j.);
     end;
     t := ok(.1,4.);

     for i := 2 to 3 do
        for j := 4 downto 1 do
        ok(.i,j.) := ok(.i,j.)*ok(.1,1.) + ok(.1,j.)*(-ok(.i,1.));

     i := 2;
     while ok(.i,2.) = 0.0 do  i := i + 1 ;
     for j := 1 to 4 do
     begin
        temp(.j.) := ok(.2,j.);
        ok(.2,j.) := ok(.i,j.);
        ok(.i,j.) := temp(.j.);
     end;

        for j := 4 downto 1 do
        ok(.3,j.) := ok(.3,j.)*ok(.2,2.) + ok(.2,j.)*(-ok(.3,2.));

     if ok(.3,3.) <> 0.0 then
     begin
        ok(.3,4.) := ok(.3,4.) / ok(.3,3.);
        ok(.2,4.) := ( ok(.2,4.) - ok(.3,4.)*ok(.2,3.)) / ok(.2,2.);
        ok(.1,4.) := ( ok(.1,4.) - ok(.2,4.)*ok(.1,2.)
                                 - ok(.3,4.)*ok(.1,3.)) / ok(.1,1.) ;
        aa := ok(.1,4.);  (* solution *)
        bb := ok(.2,4.);  (* solution *)
        cc := ok(.3,4.);  (* solution *)
    (* gauss error = t-(  aa*ok(.1,1.)+bb*ok(.1,2.)+ cc*ok(.1,3.)) *)
     end;
 end;     (* procedure gauss *)


 procedure prefix(num: integer);
 begin
             case ((num div 100) mod 10) of
                   0 : write(' ');
                   1 : write('.');
                   2 : write(':');
                   3 : write('-');
                   4 : write('+');
```

```
                    5 : write('%');
                    6 : write('*');
                    7 : write('&');
                    8 : write('?');
                    9 : write('=');
              end;
end;     (* procedure prefix *)

procedure printsub1(points:point_array);
var
   r,c,i,j,k : integer;
begin
   page;
     for k := 1 to counter do
     with subregion(.k.) do
        for i := crl+1 to crh-1 do
        for j := ccl+1 to ccr-1 do
        points(.i,j.).gp := 0;
     for r := 1 to row  do
     for c := 1 to col  do
     begin
        if c<> col  then begin
           prefix(points(.r,c.).gp);
           write((points(.r,c.).gp mod 100):2);
         end
         else begin
           prefix(points(.r,c.).gp);
           writeln((points(.r,c.).gp mod 100):2);
            writeln;
         end;
     end;
end;     (* procedure printsub1 *)


procedure printsub2(points:point_array);
var
   k,r,c,r1,r2,c1,c2 : integer;
   current : nodepointer;
begin
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
      points(.r,c.).gp := 0 ;
   for k:= 1 to counter do
   if subregion(.k.).active then

   begin
      current := subregion(.k.).list ;
      while current <> nil do begin
         r1:=current¬.r;
         c1:=current¬.c;
         if current¬.next <> nil then begin
            r2:=current¬.next¬.r;
            c2:=current¬.next¬.c;
         end
         else begin
         r2 := r1;
```

```
              c2 := c1;
              end;
              if r1 > r2 then begin r:= r2; r2:= r1; r1:= r; end;


              if c1 > c2 then begin c:= c2; c2:= c1; c1:= c; end;

              if r1=r2 then for c := c1 to c2 do
              points(.r1,c.).gp := k;
              if c1=c2 then for r := r1 to r2 do
              points(.r,c1.).gp := k;
              current := current¬.next;
           end;
        end;
          for r := 1 to row  do
          for c := 1 to col  do
          begin
             if c<> col  then begin
               prefix(points(.r,c.).gp);
               write((points(.r,c.).gp mod 100):2);
              end
              else begin
                prefix(points(.r,c.).gp);
                writeln((points(.r,c.).gp mod 100):2);
                 writeln;
              end;
          end;
     end;
        (* procedure printsub2 *)


     procedure get_nodes(var subregion: subregion_array);
     type
        direction = ( up,down,right,left );
     var
        i,j,k           : integer;
        current,head,p : nodepointer;
        temp : direction ;

        procedure get_next(var p : nodepointer;
                           var temp : direction);
        var
           i,j : integer;
           done : boolean ;


           procedure do_dir(var done:boolean ; dir:direction );
           begin
              new(p);
              p¬.r := i ;
              p¬.c := j ;
              done := true;
              case dir of
                 up    :  temp := up;
                 down  :  temp := down;
                 right :  temp := right;
          left   :  temp := left ;
       end; (* case *)
    end; (* sub_sub_ procedure do_dir *)


begin  (* sub procedure get next *)
    done := false ;
    i := p¬.r;
    j := p¬.c;
    case temp of
       right : begin
                  j := j+1;
                  while not done do begin
                     if points(.i-1,j.).gp = k
                        then do_dir(done,up);
                     if (not done and
                          (points(.i,j+1.).gp <> k))
                        then do_dir(done,down);
                     if( (subregion(.k.).list¬.r = i)
                        and (subregion(.k.).list¬.c = j)) then
                     begin
                        new(p);
                        p¬.r := i ;
                        p¬.c := j ;
                        done := true;
                     end;
                     if not done then j:= j+1;
                  end; (* while *)
               end;
       left  : begin
                  j := j-1;
                  while not done do begin
                     if points(.i+1,j.).gp = k
                        then do_dir(done,down);
                     if (not done and
                         (points(.i,j-1.).gp <> k))
                        then do_dir(done,up);
                     if not done then j:= j-1;
                  end; (* while *)
               end;
       up    : begin
                  i := i-1;
                  while not done do begin
                     if points(.i,j-1.).gp = k
                        then do_dir(done,left);
                     if (not done and
                         (points(.i-1,j.).gp <> k))
                        then do_dir(done,right);
                     if not done then i:= i-1;
                  end; (* while *)
               end;
       down  : begin
                  i := i+1;
                  while not done do begin
                     if points(.i,j+1.).gp = k
                        then do_dir(done,right);
                        if (not done and
                            (points(.i+1,j.).gp <> k))
                           then do_dir(done,left);
                        if not done then i:= i+1;
                     end; (* while *)
                  end;
        end; (* case *)
     end; (* sub procedure get_next *)


  begin   (* procedure get nodes*)
     for k := 1 to counter do
     if subregion(.k.).active then
     begin
        new(p);
        p¬.r :=subregion(.k.).crl ;
        p¬.c :=subregion(.k.).ccl ;
        while ( points(.p¬.r-1,p¬.c.).gp = k) do
           p¬.r := p¬.r-1;   (* move to the upper most row of the gp *)
        while ( points(.p¬.r,p¬.c-1.).gp = k) do
           p¬.c := p¬.c-1;   (* move to the left most col of the gp  *)

        subregion(.k.).list := p ;
        current := p ;
        temp := right ;
        repeat
           get_next(p,temp);
           current¬.next := p ;
           current := p ;
        until (subregion(.k.).list¬.r = p¬.r)
               and (subregion(.k.).list¬.c = p¬.c) ;
        p¬.next := nil;
     end;


  (*   for k := 1 to counter do
     if subregion(.k.).active then
     begin
        write('gp ',k:3,'**==>');
        current := subregion(.k.).list ;
        while current <> nil do begin
           write(current¬.r:2,',',current¬.c:2,'=>');
           current := current¬.next;
        end;
        writeln;
     end*)
  end;(* procedure get_nodes *)


  procedure makesub(rl,rh,cl,cr :integer; prev:grcuppointer);
  type
     why = (nondivisible,magnitude,std,steep);

  var
     dif,r,c : integer;
     flag    : why;
```

```
   sd,a,b,cc :real;

   procedure stopdivide(flag : why );
   var
      r,c : integer;
   begin

         counter := counter + 1;
         with subregion(.counter.) do
         begin
            crl := rl;
            crh := rh;
            ccl := cl;
            ccr := cr;
            maxmag := p¬.magmax.mag ;
            minmag := p¬.magmin.mag ;
         end;
        for c := cl to cr do
        for r := rl to rh do
           points(.r.)(.c.).gp := counter;
(*      case flag of
           nondivisible :
              write('* non divisible area!');
           magnitude    :
                write('* stop divide! by mag.');
           std          :
                write('* stop. it fits well.');
           steep        :
                write('* stop. it is too steep.');

        end;
        writeln('* area rl,rh,cl,cr=',rl:3,rh:3,cl:3,cr:3,
             ' has gpnum=',counter:3 );
*)
        p := prev ;
   end;  (*  sub procedure stop divide *)


begin    (* procedure makesub *)
     new(p);
     p¬.magmax.mag := -999;
     p¬.magmin.mag := 999;
     p¬.rl := rl;
     p¬.rh := rh;
     p¬.cl := cl;
     p¬.cr := cr;
     p¬.up := prev ;

     for c := cl to cr do
     for r := rl to rh do
     begin
        if points(.r.)(.c.).mag > p¬.magmax.mag then
        begin
               p¬.magmax.mag := points(.r.)(.c.).mag;
               p¬.magmax.r  := r;
               p¬.magmax.c  := c;
         end;
         if points(.r.)(.c.).mag < p^.magmin.mag then
         begin
                p^.magmin.mag := points(.r.)(.c.).mag;
                p^.magmin.r   := r;
                p^.magmin.c   := c;
         end;
      end;
      (*   max,min magnitude for region rl,rh,cl,cr before division *)

         getconst(points,rl,rh,cl,cr,ok,ee,ex,ey);
         gauss(ok,a,b,cc);
         sd      := sqrt(variance(a,b,cc,ee,ex,ey,ok));

         (* sd for gp before division *)

      if ((rh - rl) > 2) and ((cr - cl) > 2) then
        if (p^.magmax.mag > low_bound) and
           (p^.magmin.mag < upper_bound) then
           if sd > max_sd then
           begin
                makesub(rl , ((rh-rl) div 2) + rl,
                        cl , ((cr-cl) div 2) + cl, p);
                makesub(((rh-rl) div 2) + rl + 1 , rh,
                        cl , ((cr-cl) div 2) + cl , p);
                makesub(rl , ((rh-rl) div 2) + rl,
                       ((cr-cl) div 2) + cl + 1 , cr , p);
                makesub(((rh-rl) div 2) + rl + 1 , rh,
                       ((cr-cl) div 2) + cl + 1 , cr , p);
                p := prev ;
           end
           else stopdivide(std)
        else stopdivide(magnitude)
      else stopdivide(nondivisible);  (* smallest area *)
 end;                                (* procedure makesub *)

procedure do_combine(var pt:point_array;
                     var subregion:subregion_array;
                             var last,new : integer);
var ii,jj : integer;
    tail,current  : 0..max_group_num;
begin
   subregion(.last.).active := true;
   subregion(.new.).active := false;
   current := subregion(.last.).side ;
   if current = 0 then
   begin
         subregion(.last.).side := new;
         current := new;
   end
   else begin
            while current <> 0 do
            begin
                  tail := current ;
                  current := subregion(.current.).side ;
                                                   (* traverse *)
            end;                                        (* subregion arr *)
            subregion(.tail.).side := new;
                                                    (* to link new gp *)
            current := new;                     (* into existing link *)
         end;
   while current <> 0   do (* reassigning gp # to new region *)
   begin
      for ii := subregion(.current.).crl to
                subregion(.current.).crh do
      for jj := subregion(.current.).ccl to
                subregion(.current.).ccr do
                   pt(.ii,jj.).gp := last;
      current := subregion(.current.).side;
   end;  (* while *)
   for ii := 1 to 3 do
   for jj := 1 to 4 do
      subregion(.last.).constarr(.ii,jj.) :=
               subregion(.last.).constarr(.ii,jj.) +
               subregion(.new.).constarr(.ii,jj.);
      subregion(.last.).ee := coefs(.last.).ee +
                             subregion(.new.).ee ;
      subregion(.last.).ex := coefs(.last.).ex +
                             subregion(.new.).ex ;
      subregion(.last.).ey := coefs(.last.).ey +
                             subregion(.new.).ey ;
   with subregion(.last.) do
   begin
        gauss(constarr,a,b,c);
        sd       := sqrt(variance(a,b,c,ee,ex,ey,constarr));
   end;
end;     (* sub procedure do_combine *)


procedure combine(var pt: point_array;
                  var subregion: subregion_array);
var
   tabc_difference,i,j,last,new : integer;
   joinabc,joinsd,joined : boolean;

   procedure compareabc(var join: boolean);
   var
      temp : real ;
   begin
      temp := sqrt( sqr(subregion(.new.).a -
                           subregion(.last.).a) +
                    sqr(subregion(.new.).b -
                           subregion(.last.).b) +
                    sqr(subregion(.new.).c -
                           subregion(.last.).c) );
      if temp <= tabc_difference then join := true
      else join := false;

   end;     (* sub procedure compareabc *)

   procedure checksd(first,second: coefrec; var join: boolean);
   var
```

```
      ii,jj : integer;
   begin
      for ii := 1 to 3 do
      for jj := 1 to 4 do
         first.constarr(.ii,jj.) :=
                  first.constarr(.ii,jj.) +
                  second.constarr(.ii,jj.);
      first.ee := first.ee + second.ee ;
      first.ex := first.ex + second.ex ;
      first.ey := first.ey + second.ey ;
      with first do
      begin
          gauss(constarr,a,b,c);
          sd       := sqrt(variance(a,b,c,ee,ex,ey,constarr));
      if sd > max_sd then join := false else join := true ;
      end;
   end;   (* sub procedure check_sd_before_combine *)

begin       (* procedure combine *)
   tabc_difference := increment;
   while tabc_difference <= abc_difference do begin
   joined := false;
   for i := 2 to (row-2) do
   begin
      last := pt(.i,2.).gp ;
      for j := 2 to (col-2) do
      begin
         new := pt(.i,j+1.).gp;
         if ((i=row-2) and (j=col-2)) and (not joined) then
                  subregion(.new.).active := true;
         if (new<>last) and
            (subregion(.new.).maxmag <= upper_bound) and
            (subregion(.last.).maxmag <= upper_bound) then
         begin
            compareabc(joinabc);
            if joinabc
            then begin
               checksd( subregion(.last.),
                        subregion(.new.),joinsd );
               if joinsd then begin
                                 do_combine(pt,
                                       subregion,last,new);
                                 joined := true;
                              end
               else  begin
                        subregion(.last.).active := true;
                        joined := false ;
                        last := new ;
               end;
            end
            else  begin
                        subregion(.last.).active := true;
                        joined := false ;
                        last := new ;
            end;
         end
```

```
            else  begin
                        subregion(.last.).active := true;
                        joined := false ;
                        last := new ;
            end;
         end;
      end;    (* row combine finished *)
      for j := 2 to (row-2) do
      begin
         last := pt(.2,j.).gp ;
         for i := 2 to (col-2) do
         begin
            new := pt(.i+1,j.).gp;
            if (new<>last) and
               (subregion(.new.).minmag <= upper_bound) and
               (subregion(.last.).minmag <= upper_bound) then
            begin
               compareabc(joinabc);
               if joinabc
               then begin
                  checksd( subregion(.last.),
                           subregion(.new.),joinsd );
                  if joinsd then begin
                                    do_combine(pt,
                                       subregion,last,new);
                                    joined := true;
                                 end
                  else  begin
                           subregion(.last.).active := true;
                           joined := false ;
                           last := new ;
                  end;
               end
               else  begin
                           subregion(.last.).active := true;
                           joined := false ;
                           last := new ;
               end;
            end
            else  begin
                        subregion(.last.).active := true;
                        joined := false ;
                        last := new ;
            end;
         end;
      end;    (* column combine finished *)
      tabc_difference := tabc_difference + increment;
      end;  (* while  *)

   end;     (* procedure combine *)

   procedure cal_avg_dev(var subregion: subregion_array;
                              var points:point_array; var avg: real);
   var
      sum: real;
   i,j,gp: integer;
begin
   sum := 0.0;
   for i:= 2 to row-1 do
   for j:= 2 to col-1 do
   begin
      gp := points(.i,j.).gp;
      sum := sum + abs(( subregion(.gp.).a*i +
                         subregion(.gp.).b*j +
                         subregion(.gp.).c)
                         - points(.i,j.).el);
   end;
   avg := sum / ((row-2)*(col-2));
   writeln('average deviation = ',avg :15);
end; (* sub procedure cal_average_deviation *)


procedure writedata(points:point_array);
var
   r,c : integer;
   tempa,tempb,tempc : real;
begin
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
   begin
      tempa := subregion(.points(.r,c.).gp.).a;
      tempb := subregion(.points(.r,c.).gp.).b;
      tempc := subregion(.points(.r,c.).gp.).c;
      points(.r,c.).el := trunc (tempa*r + tempb*c + tempc);
   end;
   for c := 1 to col do
   for r := row downto 1 do
      writeln(dout1,points(.r,c.).el);
end;
   (* procedure writedata *)


procedure writetempdata(points:point_array);
var
   k,r,c,r1,r2,c1,c2 : integer;
   tempa,tempb,tempc : real;
   current : nodepointer;
begin
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
      points(.r,c.).el := 1 ;
   for k:= 1 to counter do
   if subregion(.k.).active then

   begin
      current := subregion(.k.).list ;
      while current <> nil do begin
         r1:=current¬.r;
         c1:=current¬.c;
         if current¬.next <> nil then begin
            r2:=current¬.next¬.r;
            c2:=current¬.next¬.c;
         end
         else begin
         r2 := r1;
         c2 := c1;
         end;
         if r1 > r2 then begin r:= r2; r2:= r1; r1:= r; end;


         if c1 > c2 then begin c:= c2; c2:= c1; c1:= c; end;

         tempa := subregion(.k.).a;
         tempb := subregion(.k.).b;
         tempc := subregion(.k.).c;
         if r1=r2 then for c := c1 to c2 do
         points(.r1,c.).el := trunc (tempa*r1 + tempb*c + tempc);
         if c1=c2 then for r := r1 to r2 do
         points(.r,c1.).el := trunc (tempa*r + tempb*c1 + tempc);
         current := current¬.next;
      end;
   end;
   for c := 1 to col do
   for r := row downto 1 do
     writeln(dout2,points(.r,c.).el);
end;
   (* procedure writetempdata *)


procedure check;
var
   num,r,c,i : integer;
begin
      for c:= 1 to counter do temparr(.c.) := false; (* check *)
      num := 0;
      for i := 1 to counter do
         if subregion(.i.).active then
         begin
            r:= i ;
            temparr(.r.) := true ;
            num := num +1 ;
            while r <> 0 do
            begin
               with subregion(.r.) do
                  temparr(.r.) := true;
                  r:= subregion(.r.).side ;
            end;
         end;

      for c:= 1 to counter do temparr(.c.) := false; (* check *)
      num := 0;
      for i := 1 to counter do
         if subregion(.i.).active then
         begin
            r:= i ;
            temparr(.r.) := true ;
            num := num +1 ;
            while r <> 0 do
            begin
               with subregion(.r.) do
                  temparr(.r.) := true;
                  r:= subregion(.r.).side ;
            end;
         end;

    writeln('num of gps = ',counter :3,' * lowbound=',low_bound:3,
           ' upper_bound=',upper_bound:3,'  * max_sd=',max_sd:3 );
    writeln('unexamined gps are ==>');
    c:= 0;
    for i := 1 to counter do if temparr(.i.) = true then c:= c+1
         else write('*',i:3,'*');
    writeln;
    writeln('# of gps examined =>',c:4,' * abc_difference=',
    abc_difference);
    writeln('# of combined gps =>',num :4);
end;   (* procedure check *)
```

```
(*==================================================================
                                    Main Program
 ==================================================================*)
begin
page;
   initialize ;
   new(top);
   top^.up := nil ;

   reset(data,'data ell');
   reset(datam,'data magl');
   rewrite(dout1,'data outl');   (* data after combine process *)
   rewrite(dout2,'tdata outl');  (* data of boundary *)

   for c := 1 to col do
   for r := row downto 1 do
          readln(data,points(.r.)(.c.).el);
   for c := 1 to col do
   for r := row downto 1 do
          readln(datam,points(.r.)(.c.).mag);

     makesub(2,(row-1),2,(col-1),top);
     printsubl(points); (* print the subregions created by makesub *)

     for i := 1 to counter do
        with subregion(.i.) do
        begin
           getconst(points,crl,crh,ccl,ccr,ok,ee,ex,ey);
           gauss(ok,a,b,c);

           sd       := sqrt(variance(a,b,c,ee,ex,ey,ok));
                           (* sd for region i after division *)

           constarr := ok ;
        end;
      combine(points,subregion);
      get_nodes(subregion);
      printsub2(points); (* print the subregions created by combine *)
      writedata(points) ;
             (* write modified elevation data into file 'data out#'*)
      writetempdata(points);
             (* write elevation data into file 'tempdata out#' *)
      cal_avg_dev(subregion,points,avg);
      (* check; *)
end.
```

APPENDIX B. PASCAL CODE OF STRICT BOTTOM-UP ALGORITHM

```
(*$s450000*)
(**********************************************************************
   Author   = Yee, Seung Hee.
   Date     = 3 June 1988.
   Input    = 1. Elevation data acquired from Fort. Hunter Liggett
                 data base (40 by 40 square).
              2. file of magnitude of gradient of the above elevation
                 data created by using the program in Appendix 3.
              3. file of curvature of points created by using the
                 program in Appendix 3.
   Output   = 1. Elevation data of the model created by this program.
              2. Elevation data of the boundary points, other points
                 are given the value one so that when we display
                 this model in three-dimensional pictures, we can
                 only see the boundaries of the patches.
   Computer = Waterloo Pascal on IBM 370/3033AP,
              Naval Postgraduate School.
   Description =
              This program is the Pascal implementation of the
              Joint top-down and bottom-up terrain modeling.
 **********************************************************************)

PROGRAM Bottom_up(input,output);
const
   low_bound = 0.1;    (* magnitude < 10 % then level slope  *)
                       (* magnitude > 50 % then unsafe slope *)
   up_bound = 0.6;     (*           else safe slope          *)
   magnitude_ratio = 0.2; (* ratio of magnitude :
                                  dif of two mags / old mag *)
   abc_comparison_const =20;
   max_group_num = 400;
   row = 40;
   col = 40;
type
   elrange = 1..10000;
   mag_type = (level,safe,unsafe);
   point   = record
               r,c: integer;
             end;
   nodepointer = ¬node;
   node  =  record
               r,c : integer;
               next : nodepointer;
            end;
   point_rec = record
                 el : elrange ;
                 mag : real;
                 magtype: mag_type;
                 cur: char;
                 gp : 0..max_group_num;
                 ptnext : point;
                 end;
      point_array = array (.1..row,1..col.) of point_rec;
      polynomial_array = array (.1..3,1..4.) of real;
      coef_record = record
                        start : point;
                        upper_left : point;
                        a,b,c : real ;
                        sd   : real ;
                        ee,ex,ey : real;
                        side : 0..max_group_num;
                        poly_array : polynomial_array;
                        active : boolean;
                        list  : nodepointer ;
                        numpts : integer;
                        avgmag: real;
                    end;
      coef_array = array (.1..max_group_num.) of coef_record ;
   var
      points : point_array ;
      subregion  : coef_array ;
      ok     : polynomial_array;
      data,datam,datac,dout1,dout2,dout3,dout4  : text;
      counter,r,c,i,num : integer;
      tempmag  : real;

   function variance (a,b,c,ee,ex,ey: real; ok: polynomial_array): real;
   begin
           variance := (ee-(2*a*ex)-(2*b*ey)-(2*c*ok(.3,4.))
                          +(a*a*ok(.1,1.))+(2*a*b*ok(.2,1.))
                          +(2*b*c*ok(.2,3.))+(2*a*c*ok(.1,3.))
                          +(b*b*ok(.2,2.))+(c*c*ok(.3,3.)))
                          /(ok(.3,3.)-1);
   end;
   procedure initialize;
   var
      r,c :integer;
   begin
         counter := 0 ;
         for c := 1 to col do
         for r := 1 to row do
         begin
            points(.r,c.).gp := 0 ;
            points(.r,c.).ptnext.r := 0 ;
            points(.r,c.).ptnext.c := 0 ;
         end;
         for r := 1 to max_group_num do
         begin
             subregion(.r.).a := -999;
             subregion(.r.).b := -999;
             subregion(.r.).c := -999;
             subregion(.r.).active := true;
             subregion(.r.).side := 0;
             subregion(.r.).sd := -999;

         end;
   end;             (* procedure initialize *)
```

```
procedure  getconst( r,c:integer;
                     var arr:polynomial_array; var ee,ex,ey :real);
begin
             arr(.1,1.):=arr(.1,1.)+(r*r);
             arr(.1,2.):=arr(.1,2.)+(c*r);
             arr(.1,3.):=arr(.1,3.)+ r;
             arr(.1,4.):=arr(.1,4.)+(points(.r,c.).el*r);

             arr(.2,1.):=arr(.2,1.)+(r*c);
             arr(.2,2.):=arr(.2,2.)+(c*c);
             arr(.2,3.):=arr(.2,3.)+ c;
             arr(.2,4.):=arr(.2,4.)+(points(.r,c.).el*c);

             arr(.3,1.):=arr(.3,1.)+r;
             arr(.3,2.):=arr(.3,2.)+c;
             arr(.3,3.):=arr(.3,3.)+1;
             arr(.3,4.):=arr(.3,4.)+points(.r,c.).el;

             ee := ee + sqr(points(.r,c.).el);
             ex := ex + (points(.r,c.).el * r );
             ey := ey + (points(.r,c.).el * c );
end;          (* procedure get const *)

procedure gauss(ok:polynomial_array; var aa,bb,cc:real);

var
   i,j : integer;
   temp : array (.1..4.) of real;
   t : real;                          (* error checking purpose *)
begin

   i := 1;
   while ok(.i,1.) = 0.0 do  i := i + 1 ; (* row rotation *)
   for j := 1 to 4 do
   begin
      temp(.j.) := ok(.1,j.);
      ok(.1,j.) := ok(.i,j.);
      ok(.i,j.) := temp(.j.);
   end;
   t := ok(.1,4.);

   for i := 2 to 3 do
      for j := 4 downto 1 do
      ok(.i,j.) := ok(.i,j.)*ok(.1,1.) + ok(.1,j.)*(-ok(.i,1.));

   i := 2;
   while ok(.i,2.) = 0.0 do  i := i + 1 ;
   for j := 1 to 4 do
   begin
      temp(.j.) := ok(.2,j.);
      ok(.2,j.) := ok(.i,j.);
      ok(.i,j.) := temp(.j.);
   end;

      for j := 4 downto 1 do
```

```
         ok(.3,j.) := ok(.3,j.)*ok(.2,2.) + ok(.2,j.)*(-ok(.3,2.));

      if ok(.3,3.) <> 0.0 then
      begin
         ok(.3,4.) := ok(.3,4.) / ok(.3,3.);
         ok(.2,4.) := ( ok(.2,4.) - ok(.3,4.)*ok(.2,3.)) / ok(.2,2.);
         ok(.1,4.) := ( ok(.1,4.) - ok(.2,4.)*ok(.1,2.)
                                  - ok(.3,4.)*ok(.1,3.)) / ok(.1,1.) ;
         aa := ok(.1,4.);  (* solution *)
         bb := ok(.2,4.);  (* solution *)
         cc := ok(.3,4.);  (* solution *)
  (*writeln('gauss error =',
                 t-(  aa*ok(.1,1.)+bb*ok(.1,2.)+ cc*ok(.1,3.)))*)
      end;
  end;      (* procedure gauss *)

  procedure prefix(num: integer);
  begin
             case ((num div 100) mod 10) of
                   0 : write(' ');
                   1 : write('.');
                   2 : write(':');
                   3 : write('-');
                   4 : write('+');
                   5 : write('%');
                   6 : write('*');
                   7 : write('&');
                   8 : write('?');
                   9 : write('=');
             end;
 end;      (* procedure prefix *)

 procedure printsub(points: point_array);
 var
    r,c : integer;
 begin
 page;
      for r := 1 to row do
      for c := 1 to col do
      begin
         if c<> col then begin
            prefix(points(.r,c.).gp);
            write((points(.r,c.).gp mod 100):2);
          end
          else begin
            prefix(points(.r,c.).gp);
            writeln((points(.r,c.).gp mod 100):2);
            writeln;
          end;
      end;
 end;      (* procedure printsub *)


 procedure printsub2(points: point_array);
 var
    k,r,c,r1,r2,c1,c2 : integer;
```

```
   current : nodepointer;
begin
page;
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
      points(.r,c.).gp := 0 ;
   for k:= 1 to counter do
   if subregion(.k.).active then
   begin
      current := subregion(.k.).list ;
      while current <> nil do
      begin
         r1:=current¬.r;
         c1:=current¬.c;
         if current¬.next <> nil then begin
            r2:=current¬.next¬.r;
            c2:=current¬.next¬.c;
         end
         else begin
            r2 := r1;
            c2 := c1;
         end;
         if r1 > r2 then begin r:= r2; r2:= r1; r1:= r; end;
         if c1 > c2 then begin c:= c2; c2:= c1; c1:= c; end;

         if r1=r2 then for c := c1 to c2 do
         points(.r1,c.).gp := k;
         if c1=c2 then for r := r1 to r2 do
         points(.r,c1.).gp := k;
         current := current¬.next;
      end;
   end;
     for r := 1 to row  do
     for c := 1 to col  do
     begin
        if c<> col  then begin
           prefix(points(.r,c.).gp);
           write((points(.r,c.).gp mod 100):2);
         end
         else begin
           prefix(points(.r,c.).gp);
           writeln((points(.r,c.).gp mod 100):2);
            writeln;
         end;
     end;
end;
   (* procedure printsub2 *)

procedure printmagcur;
var
   r,c : integer;
begin
(* page*)
     for r := 1 to row  do
     for c := 1 to col  do
     begin
```

```
         if c<> col then begin
            case points(.r,c.).magtype of
               level :write('l',points(.r,c.).cur,' ');
               safe  :write('s',points(.r,c.).cur,' ');
               unsafe  :write('u',points(.r,c.).cur,' ');
           end; (* case *)
         end
         else begin
            case points(.r,c.).magtype of
               level :writeln('l',points(.r,c.).cur,' ');
               safe  :writeln('s',points(.r,c.).cur,' ');
               unsafe:writeln('u',points(.r,c.).cur,' ');
            end;
            writeln;
         end;
      end;
end;     (* procedure printmagcur *)

procedure makeone_rowl(var point1,point2 : point);
begin
         points(.point1.r,point1.c.).ptnext.r := point2.r;
         points(.point1.r,point1.c.).ptnext.c := point2.c;
         points(.point2.r,point2.c.).gp :=
                     points(.point1.r,point1.c.).gp;
         with subregion(.counter.) do
              numpts := numpts +1;



end;     (* sub procedure make one_1 *)


procedure makeone_row2(var point1,point2 : point);
begin
         points(.point1.r,point1.c.).ptnext.r := point2.r;
         points(.point1.r,point1.c.).ptnext.c := point2.c;
         points(.point2.r,point2.c.).gp :=
                     points(.point1.r,point1.c.).gp;
         with subregion(.counter.) do
         begin
            numpts := numpts +1;
            avgmag := avgmag*(numpts-1)/numpts +
                       points(.point2.r,point2.c.).mag/numpts;
         end;

end;     (* sub procedure make one_2 *)


procedure rowlink;
var
   i,j           : integer;
   point1,point2 : point;
begin        (* sub procedure rowlink *)
   counter := 0;
   for i := 2 to (row-1) do
   begin
```

```
      counter := counter + 1 ;
      points(.i,2.).gp := counter;
      subregion(.counter.).start.r := i;
      subregion(.counter.).start.c := 2;
      subregion(.counter.).upper_left :=
                   subregion(.counter.).start ;
      subregion(.counter.).avgmag  := points(.i,2.).mag;
      subregion(.counter.).numpts  := 1 ;
      for j := 2 to (col-2) do
      begin
         point1.r := i;
         point1.c := j;
         point2.r := i;
         point2.c := j+1;
         if ((points(.i,j+1.).magtype <> safe) and
            (points(.i,j.).magtype = points(.i,j+1.).magtype)) then
            makeone_row1(point1,point2)
                (* (level level) or (unsafe unsafe) *)
         else if ((points(.i,j.).magtype = safe) and
                  (points(.i,j+1.).magtype = safe)) and
                 (points(.i,j.).cur = points(.i,j+1.).cur) then
                 if (abs(points(.i,j+1.).mag-
                     subregion(.counter.).avgmag)/
                     subregion(.counter.).avgmag) <
                     magnitude_ratio then
                     makeone_row2(point1,point2)
                 else begin
                       counter := counter +1 ;
                       points(.i,j+1.).gp := counter;
                       subregion(.counter.).start := point2;
                       subregion(.counter.).upper_left := point2 ;
                       subregion(.counter.).avgmag :=
                                    points(.i,j+1.).mag;
                       subregion(.counter.).numpts := 1 ;
                       point1 := point2;
                 end

              else begin
                    counter := counter +1 ;
                    points(.i,j+1.).gp := counter;
                    subregion(.counter.).start := point2;
                    subregion(.counter.).upper_left := point2 ;
                    subregion(.counter.).avgmag :=
                                     points(.i,j+1.).mag;
                    subregion(.counter.).numpts := 1 ;
                    point1 := point2;
              end;
      end;
   end;
end;        (* sub procedure row link *)


procedure makeone_col(last,new : point);

var
   ii,jj : integer;
   tail,current  : point;

begin
   subregion(.points(.new.r,new.c.).gp.).active := false ;
   with subregion(.points(.last.r,last.c.).gp.) do
   begin
      numpts:=numpts +
              subregion(.points(.new.r,new.c.).gp.).numpts;
      avgmag:=avgmag*(numpts-
              subregion(.points(.new.r,new.c.).gp.).numpts)/
              numpts +
              subregion(.points(.new.r,new.c.).gp.).avgmag*
              subregion(.points(.new.r,new.c.).gp.).numpts/numpts;
   end;       (* reassigning number of points and average magnitude  *)

   if (subregion(.points(.new.r,new.c.).gp.).upper_left.r +
       subregion(.points(.new.r,new.c.).gp.).upper_left.c) <
      (subregion(.points(.last.r,last.c.).gp.).upper_left.r +
       subregion(.points(.last.r,last.c.).gp.).upper_left.c) then
   subregion(.points(.last.r,last.c.).gp.).upper_left :=
       subregion(.points(.new.r,new.c.).gp.).upper_left ;

   current := points(.last.r,last.c.).ptnext ;
   if current.r = 0 then          (* or current.c = 0 *)
   begin
         points(.last.r,last.c.).ptnext  :=
             subregion(.points(.new.r,new.c.).gp.).start ;
         current :=  subregion(.points(.new.r,new.c.).gp.).start ;
   end
   else begin
           while current.r <> 0 do
           begin
                 tail := current;
                 current := points(.current.r,current.c.).ptnext;
           end;                 (*  traverse ptnext of lastpt link
                                         to the end of link *)
           points(.tail.r,tail.c.).ptnext:=   (* now link the tail
                                                points to the head of*)
              subregion(.points(.new.r,new.c.).gp.).start ;
                             (* newpt link *)
           current :=                      (* into existing link   *)
              subregion(.points(.new.r,new.c.).gp.).start ;
        end;
   while current.r <> 0 do (* reassigning gp # to new pts *)
   begin
      points(.current.r,current.c.).gp := points(.last.r,last.c.).gp;
      current := points(.current.r,current.c.).ptnext;
   end;  (* while *)
end;     (* procedure makeone_col *)


procedure collink;
var
   i,j : integer;
   point1,point2 : point;
begin      (* sub procedure col link *)
```

```
   for j := 2 to (col-1) do
   begin
      for i := 2 to (row-2) do
      begin
         point1.r := i;
         point1.c := j;
         point2.r := i+1;
         point2.c := j;
         if (points(.i+1,j.).gp <> points(.i,j.).gp) then
            if ((points(.i+1,j.).magtype <> safe) and
                (points(.i,j.).magtype = points(.i+1,j.).magtype)) then
                   makeone_col(point1,point2)
                  (* (level,level) or (unsafe,unsafe) *)
            else if ((points(.i,j.).magtype = safe) and
                     (points(.i+1,j.).magtype = safe)) then
               if (points(.i,j.).cur = points(.i+1,j.).cur) then
                  if abs(subregion(.points(.i+1,j.).gp.).avgmag-
                         subregion(.points(.i,j.).gp.).avgmag)/
                         subregion(.points(.i,j.).gp.).avgmag <
                         magnitude_ratio
                  then
                     makeone_col(point1,point2);
         point1 := point2;
      end;
   end;
end; (* sub procedure collink *)

procedure do_combine(last,new : point);

var
   ii,jj : integer;
   tail,current  : point;

begin
   subregion(.points(.new.r,new.c.).gp.).active := false ;
   with subregion(.points(.last.r,last.c.).gp.) do
   begin
      numpts:=numpts +
              subregion(.points(.new.r,new.c.).gp.).numpts;
   end;       (* reassigning number of points *)

   if (subregion(.points(.new.r,new.c.).gp.).upper_left.r +
       subregion(.points(.new.r,new.c.).gp.).upper_left.c) <
      (subregion(.points(.last.r,last.c.).gp.).upper_left.r +
       subregion(.points(.last.r,last.c.).gp.).upper_left.c) then
   subregion(.points(.last.r,last.c.).gp.).upper_left :=
       subregion(.points(.new.r,new.c.).gp.).upper_left ;


   current := points(.last.r,last.c.).ptnext ;
   if current.r = 0 then          (* or current.c = 0 *)
   begin
         points(.last.r,last.c.).ptnext  :=
             subregion(.points(.new.r,new.c.).gp.).start ;
         current :=  subregion(.points(.new.r,new.c.).gp.).start ;
   end
```

```
    else begin
            while current.r <> 0 do
            begin
                  tail := current;
                  current := points(.current.r,current.c.).ptnext;
            end;                (*  traverse ptnext to link the head *)
            points(.tail.r,tail.c.).ptnext:=               (* of the *)
               subregion(.points(.new.r,new.c.).gp.).start ;
                          (* new point *)
            current :=                          (* into existing link   *)
               subregion(.points(.new.r,new.c.`.gp.).start ;
        end;
    while current.r <> 0 do (* reassigning gp # to new pts *)
    begin
       points(.current.r,current.c.).gp := points(.last.r,last.c.).gp;
       current := points(.current.r,current.c.).ptnext;
    end;  (* while *)


    for ii := 1 to 3 do
    for jj := 1 to 4 do
    subregion(.points(.last.r,last.c.).gp.).poly_array(.ii,jj.) :=
         subregion(.points(.last.r,last.c.).gp.).poly_array(.ii,jj.)+
         subregion(.points(.new.r,new.c.).gp.).poly_array(.ii,jj.);
    subregion(.points(.last.r,last.c.).gp.).ee :=
         subregion(.points(.last.r,last.c.).gp.).ee +
         subregion(.points(.new.r,new.c.).gp.).ee ;
    subregion(.points(.last.r,last.c.).gp.).ex :=
         subregion(.points(.last.r,last.c.).gp.).ex +
         subregion(.points(.new.r,new.c.).gp.).ex ;
    subregion(.points(.last.r,last.c.).gp.).ey :=
         subregion(.points(.last.r,last.c.).gp.).ey +
         subregion(.points(.new.r,new.c.).gp.).ey ;
    with subregion(.points(.last.r,last.c.).gp.) do
    begin
         sd       := sqrt(variance(a,b,c,ee,ex,ey,poly_array));
    end;
end;      (* procedure do_combine *)


procedure combine(var pt: point_array;
                  var subregion:coef_array);
var
   i,j : integer;
   last,new : point;
   joinabc,joinsd,joined : boolean;

   procedure compareabc(var join:boolean);
   var
      temp : real ;
   begin
      temp := sqrt( sqr(subregion(.points(.new.r,new.c.).gp.).a -
                     subregion(.points(.last.r,last.c.).gp.).a) +
                    sqr(subregion(.points(.new.r,new.c.).gp.).b -
                     subregion(.points(.last.r,last.c.).gp.).b) +
                    sqr(subregion(.points(.new.r,new.c.).gp.).c -
                        subregion(.points(.last.r,last.c.).gp.).c) );
           if temp <= abc_comparison_const then join := true
           else join := false;

        end;     (* sub procedure compareabc *)


     begin        (* procedure combine *)
        joined := false;
        for i := 2 to (row-2) do
        begin
           last.r := i;
           last.c := 2;
           for j := 2 to (col-2) do
           begin
              new.r := i;
              new.c := j+1;
              if ((i=row-2) and (j=col-2)) and (not joined) then
                        subregion(.points(.new.r,new.c.).gp.).active :=
                                                                   true;
              if (points(.new.r,new.c.).gp <> points(.last.r,last.c.).gp)
              then begin
                 compareabc(joinabc);
                 if joinabc
                 then begin
                        do_combine(last,new);
                        joined := true;
                      end
                 else  begin
                         subregion(.points(.last.r,last.c.).gp.).active :=
                                                                   true;
                         joined := false ;
                         last := new ;
                       end;    (* joinabc *)
              end
              else  begin
                      subregion(.points(.last.r,last.c.).gp.).active :=
                                                                   true;
                      joined := false ;
                      last := new ;
                    end;    (* last <> new *)
           end;(* end j *)
        end;    (* end i *)(* row combine finished *)
        for j := 2 to (row-2) do
        begin
           last.r := 2;
           last.c := j;
           for i := 2 to (col-2) do
           begin
              new.r := i+1;
              new.c := j ;
              if (points(.new.r,new.c.).gp <> points(.last.r,last.c.).gp)
              then begin
                 compareabc(joinabc);
                 if joinabc
                 then begin
```

```
                        do_combine(last,new);
                        joined := true;
                     end
                 else  begin
                        subregion(.points(.last.r,last.c.).gp.).active :=
                                                                   true;
                        joined := false ;
                        last := new ;
                      end;     (* join abc *)
             end
             else  begin
                      subregion(.points(.last.r,last.c.).gp.).active :=
                                                                   true;
                      joined := false ;
                      last := new ;
                    end;    (* last <> new *)
           end;
       end;    (* column combine finished *)

   end;     (* procedure combine *)



   procedure get_boundaries(var subregion: coef_array);
   type
      direction = ( up,down,right,left );
   var
      i,j,k,r,c       : integer;
      current,head,p : nodepointer;
      temp : direction ;

      procedure get_next(var p : nodepointer;
                         var temp : direction);
      var
         i,j : integer;
         done : boolean ;


         procedure do_dir(var done:boolean ; dir:direction );
         begin
            new(p);
            p^.r := i ;
            p^.c := j ;
            done := true;
            case dir of
               up    :  temp := up;
               down  :  temp := down;
               right :  temp := right;
               left  :  temp := left ;
            end; (* case *)
         end; (* sub_sub_ procedure do_dir *)


      begin  (* sub procedure get next *)
         done := false ;
         i := p^.r;
```

```
j := p¬.c;
case temp of
   right : begin
              j := j+1;
              while not done do begin
                 if points(.i-1,j.).gp = k
                    then do_dir(done,up)   (* go up and done *)
                 else if( (subregion(.k.).list¬.r = i)
                    and (subregion(.k.).list¬.c = j)) then
                 begin                    (* return to org point *)
                    new(p);
                    p¬.r := i ;
                    p¬.c := j ;
                    done := true;
                 end
                 else if (points(.i,j+1.).gp = k)
                    then j:= j+1           (* go right *)
                 else if (points(.i+1,j.).gp=k )
                    then do_dir(done,down) (* go down and done *)
                 else do_dir(done,left);    (* go back and done *)
              end; (* while *)
           end;
   left  : begin
              j := j-1;
              while not done do begin
                 if points(.i+1,j.).gp = k
                    then do_dir(done,down) (* go down and done*)
                 else if (points(.i,j-1.).gp = k)
                    then j := j-1              (* go left *)
                 else if (points(.i-1,j.).gp = k)
                    then do_dir(done,up)   (* go up and done *)
                 else do_dir(done,right); (* go back and done *)
              end; (* while *)
           end;
   up    : begin
              i := i-1;
              while not done do begin
                 if points(.i,j-1.).gp = k
                    then do_dir(done,left) (* go left and done *)
                 else if (points(.i-1,j.).gp = k)
                    then i := i-1          (* go up  *)
                 else if (points(.i,j+i.).gp = k )
                    then do_dir(done,right)(* go right and done*)
                 else do_dir(done,down);   (* go down and done *)
              end; (* while *)
           end;
   down  : begin
              i := i+1;
              while not done do begin
                 if points(.i,j+1.).gp = k
                    then do_dir(done,right)(* go right and done*)
                 else if points(.i+1,j.).gp = k
                    then i := i+1          (* go down *)
                 else if points(.i,j-1.).gp = k
                    then do_dir(done,left)(* go left and done*)
                 else do_dir(done,up);     (* go up and done *)
```

```
                  end; (* while *)
                end;
         end; (* case *)
       end; (* sub procedure get_next *)


     begin   (* procedure get boundaries *)
       for k := 1 to counter do
       if subregion(.k.).active then
       begin
         new(p);
         p^.r :=subregion(.k.).upper_left.r ;
         p^.c :=subregion(.k.).upper_left.c ;
         while ( points(.p^.r-1,p^.c.).gp = k) do
           p^.r := p^.r-1;   (* move to the upper most row of the gp *)
         while ( points(.p^.r,p^.c-1.).gp = k) do
           p^.c := p^.c-1;   (* move to the left most col of the gp  *)

         subregion(.k.).list := p ;
         if points(.p^.r,p^.c+1.).gp = k then
         begin                (* starting toward right groups *)
           current := p ;
           temp := right ;
           repeat
             get_next(p,temp);
             current^.next := p ;
             current := p ;
           until (subregion(.k.).list^.r = p^.r)
                  and (subregion(.k.).list^.c = p^.c);
           if (temp=down) and (points(.p^.r+1,p^.c.).gp=k) then
           begin                 (* right and down points gp *)
             current := p ;      (* hinged on start point    *)
             temp := down ;
             repeat
               get_next(p,temp);
               current^.next := p ;
               current := p ;
             until (subregion(.k.).list^.r = p^.r)
                    and (subregion(.k.).list^.c = p^.c);
             p^.next := nil;
           end
           else
             p^.next := nil; (* right but not down gp *)

         end
         else if points(.p^.r+1,p^.c.).gp = k then
         begin              (* starting toward down group *)
           current := p ;
           temp := down ;
           repeat
             get_next(p,temp);
             current^.next := p ;
             current := p ;
           until (subregion(.k.).list^.r = p^.r)
                  and (subregion(.k.).list^.c = p^.c);
           p^.next := nil;
```

```
      end
      else
         p¬.next := nil; (* one point gp *)
   end;
(*=====================================================================
   the following segment of code may be used for checking purpose
   for k := 1 to counter do
   if subregion(.k.).active then
   begin
      write('Boundary of group ',k:3,'is ==>');
      current := subregion(.k.).list;
      while current <> nil do
      begin
         write(current¬.r:2,',',current¬.c:2,'=>');
         current := current¬.next;
      end;
      writeln;
   end;
=====================================================================*)
end;  (* procedure get_boundaries *)

procedure calc_coeffs;
var
   k,i,j,r,c,gpnum,ptnum : integer;
   t,t1  : point;
   rowchanged,colchanged: boolean;
begin
      gpnum := 0;
      ptnum := 0;
      for k := 1 to counter do
         if subregion(.k.).active then
         begin
            rowchanged := false;
            colchanged := false;
            for i := 1 to 3 do
            for j := 1 to 4 do
               subregion(.k.).poly_array(.i,j.) := 0;
            subregion(.k.).ee := 0;
            subregion(.k.).ex := 0;
            subregion(.k.).ey := 0;
            ptnum := ptnum +subregion(.k.).numpts ;
                                  (* num of linked pt *)
            gpnum := gpnum +1 ;                (* num of active gp *)
            t := subregion(.k.).start;
            while t.r <> 0 do
            begin
                  t1 := t;
                  with subregion(.k.) do
                     getconst( t.r, t.c,poly_array,ee,ex,ey);
                  t:= points(.t.r,t.c.).ptnext ;
                  if (t.r<>0) and (t.r<>t1.r) then
                     rowchanged := true;
                  if (t.c<>0) and (t.c<>t1.c) then
                     colchanged := true;
            end;
            with subregion(.k.) do
            begin
               if rowchanged and colchanged then
                  gauss(poly_array,a,b,c)         (* normal points *)
               else if not(rowchanged) and not(colchanged) then
               begin
                  a := 0;
                  b := 0;
                  c := poly_array(.3,4.);
               end                               (* one point gp *)
               else if not(rowchanged) and colchanged then
               begin                      (* horizontal line group *)
                  a := 0;
                  for j:= 4 downto 2 do
                  poly_array(.3,j.) :=
                        poly_array(.3,j.) * poly_array(.2,2.) +
                        poly_array(.2,j.) * (-poly_array(.3,2.));
                  poly_array(.3,4.) :=
                        poly_array(.3,4.) /
                        poly_array(.3,3.);
                  poly_array(.2,4.) := (poly_array(.2,4.)   -
                                        poly_array(.3,4.) *
                                        poly_array(.2,3.))  /
                                        poly_array(.2,2.);
                  b := poly_array(.2,4.);
                  c := poly_array(.3,4.);
               end
               else if rowchanged and not(colchanged) then
               begin             (* vertical line group *)
                  b := 0;
                  for j:= 4 downto 1 do
                  poly_array(.3,j.) :=
                        poly_array(.3,j.)*
                        poly_array(.1,1.) +
                        poly_array(.1,j.)*
                        (-poly_array(.3,1.));
                  poly_array(.3,4.) :=
                        poly_array(.3,4.) /
                        poly_array(.3,3.);
                  poly_array(.1,4.) := (poly_array(.1,4.)-
                        poly_array(.3,4.)*
                        poly_array(.1,3.))/
                        poly_array(.1,1.);
                  a := poly_array(.1,4.);
                  c := poly_array(.3,4.);
               end;
(*             writeln(' a,b,c,=',a:10,' ',b:10,' ',c:10) *)
            end;      (*  with subregion(.k.)    *)
         end;         (* if subregion(.k.).active *)

   writeln('active pts=',ptnum:5,' * active gpnum =',gpnum:3);
end; (* procedure calc_coeffs *)


procedure write_planar_data(points:point_array);
var
   r,c : integer;
   tempa,tempb,tempc : real;
begin
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
   begin
      tempa := subregion(.points(.r,c.).gp.).a;
      tempb := subregion(.points(.r,c.).gp.).b;
      tempc := subregion(.points(.r,c.).gp.).c;
      points(.r,c.).el := trunc (tempa*r + tempb*c + tempc);
   end;
   for c := 1 to col do
   for r := row downto 1 do
       writeln(dout1,points(.r,c.).el);
end;
   (* procedure write_planar_data *)


procedure write_boundary_data(points:point_array);
var
   k,r,c,r1,r2,c1,c2 : integer;
   tempa,tempb,tempc : real;
   current : nodepointer;
begin
   for r:= 2 to row-1 do
   for c:= 2 to col-1 do
      points(.r,c.).el := 1 ;
   for k:= 1 to counter do
   if subregion(.k.).active then
   begin
      current := subregion(.k.).list ;
      while current <> nil do
      begin
         r1:=current¬.r;
         c1:=current¬.c;
         if current¬.next <> nil then begin
            r2:=current¬.next¬.r;
            c2:=current¬.next¬.c;
         end
         else begin
            r2 := r1;
            c2 := c1;
         end;
         if r1 > r2 then begin r:= r2; r2:= r1; r1:= r; end;
         if c1 > c2 then begin c:= c2; c2:= c1; c1:= c; end;

         tempa := subregion(.k.).a;
         tempb := subregion(.k.).b;
         tempc := subregion(.k.).c;
         if r1=r2 then for c := c1 to c2 do
         points(.r1,c.).el := trunc (tempa*r1 + tempb*c + tempc);
         if c1=c2 then for r := r1 to r2 do
         points(.r,c1.).el := trunc (tempa*r + tempb*c1 + tempc);
         current := current¬.next;
      end;
   end;
   for c := 1 to col do
   for r := row downto 1 do
       writeln(dout2,points(.r,c.).el);
end;  (* write_boundary_data *)
   (* procedure write_boundary_data *)
```

```
(*=================================================================
                         Main Program
 ================================================================*)
begin
page;
   initialize ;

   reset(data,'data ell');
   reset(datam,'data magl');
   reset(datac,'data curl');
   rewrite(doutl,'data outl');   (* data after combine process *)
   rewrite(dout2,'tdata outl');  (* data of boundary *)

   for c := 1 to col do
   for r := row downto 1 do
        readln(data,points(.r,c.).el);
   for c := 1 to col do
   for r := row downto 1 do
   begin
        readln(datam,tempmag);
        if tempmag < low_bound then
           points(.r,c.).magtype := level
        else if tempmag > low_bound then
           points(.r,c.).magtype := safe
        else   points(.r,c.).magtype := unsafe;
        points(.r,c.).mag := tempmag;
   end;
   for c := 1 to col do
   for r := row downto 1 do
        readln(datac,points(.r,c.).cur);

(*printmagcur*)    (* print magnitude and curvature *)
rowlink;        (* link points in row and make subregions *)
printsub(points);(* print subregions created by rowlink *)
collink;        (* link points in row and column and make subregions *)
calc_coeffs;     (* calculate plane expressions for the
                    subregions created *)
combine(points,subregion);
                        (* combine adjacent regions if they
                         have similar plane expressions *)
calc_coeffs;            (* calculate plane expressions for the
                         final subregions *)
get_boundaries(subregion);
                       (* get boundaries of subregions in linked list *)
printsub2(points);     (* print subregions created by combine *)
write_planar_data(points);  (* write out the elevation of the model *)
write_boundary_data(points);(* write out the elevation of
                               the boundaries of the subregions *)
end.
```

APPENDIX C. PASCAL CODE FOR SMOOTHING ELEVATION DATA AND CALCULATING GRADIENT

```
(*Ss300000*)
program calc_slope_curv_smooth(input,output);

(*****************************************************************************
   Author    = Yee, Seung Hee.
   Date      = 3 June 1988.
   Input     = 1. Elevation data acquired from Fort. Hunter Liggett
                  data base (40 by 40 square).
   Output    = 1. File of magnitude of gradient of the points.
               2. FIle of curvature of the points (optional).
               3. FIle of smoothed elevation data (optional).
   Computer = Waterloo Pascal on IBM 370/3033AP,
              Naval Postgraduate School.
   Description =
   This program contains two major procedures:
   calc_slope_curvature and smoothing.
   The procedure calc_slope_curvature calculates the slope
   and curvature of each points, the procedure smoothing processes the
   elevation data to get smoothed elevation data.
   Original elevation data is represented in feet and there are 40*40
   data cells in one window. The distance between each cell is12.5
   meters , so it needs to be converted to get a metric system.
 *****************************************************************************)

const
     distance_unit = 1;    (* 3.28084*12.5 = 41.0105 *)
     row = 3;
     col = 3;
     curv_thresh = 0.05; (* threshold for eigenvalues of biquadratic *)
     upbound = 0.6;  (* slope > 60 % then unsafe slope *)

     lobound = 0.1;   (* slope < 10 % then level slope *)
type
    elrange = 0..10000;
    pointrec = record
                  el : elrange ;
                  slope: real;
                  cur: char;
               end;
    point_array=array (.1..row,1..col.) of pointrec ;
var
   points,spoints : point_array ;
   data,outslope,outcur,smoutel,smoutslope,smoutcur   : text;
   counter,r,c,i,num: integer;
   k2,k3,k4,k5,k6,slope,lam1,lam2: real;


procedure init(var pt:point_array);
var
```

```
   r,c,i,j: integer;
begin
    for r := 1 to row do
    begin
       pt(.r,1.).slope := 999;
       pt(.r,col.).slope := 999;
       pt(.r,1.).cur := 'u';
       pt(.r,col.).cur := 'u';
    end;
    for c := 1 to col do
    begin
       pt(.1,c.).slope := 999;
       pt(.row,c.).slope := 999;
       pt(.1,c.).cur := 'u';
       pt(.row,c.).cur := 'u';
    end;

end;
procedure calc_slope_curvature(var pt:point_array);

var
   r,c,i,j: integer;
   templ,temp2,termx,termy,dtermx,dtermy,
   templam  ,termxx,termyy,termxy,term :real;
begin
     for r:= 2 to row-1 do
     for c:= 2 to col-1 do
     begin
         term := 0.0;
         termx := 0.0;
         termy := 0.0;
         termxx := 0.0;
         termyy := 0.0;
         termxy := 0.0;

         for i :=-1 to 1 do
         for j :=-1 to 1 do
         begin
            term := term + pt(.r+i,c+j.).el;
            termx := termx + j*pt(.r+i,c+j.).el;
            termy := termy + i*pt(.r+i,c+j.).el;
            termxx := termxx + j*j*pt(.r+i,c+j.).el;
            termyy := termyy + i*i*pt(.r+i,c+j.).el;
            termxy := termxy + i*j*pt(.r+i,c+j.).el;
         end;

         k2 := termx/(6*distance_unit);
         k3 := termy/(6*distance_unit);
         k4 := (termxx/2 - term/3)/distance_unit;
         k5 := termxy/(4*distance_unit) ;
         k6 := (termyy/2 - term/3)/distance_unit;
         lam1 := ((2*k4+2*k6) +
            sqrt(abs(sqr(-2*k4-2*k6)-4*(2*k4*2*k6-sqr(k5)))))/2;
         lam2 := ((2*k4+2*k6) -
            sqrt(abs(sqr(-2*k4-2*k6)-4*(2*k4*2*k6-sqr(k5)))))/2;
         if abs(lam1) < abs(lam2) then
         begin
            templam := lam1;
            lam1 := lam2;
            lam2 := templam;
         end;   (* eigenvalues of bigger slopenitude becomes lam1 *)
         slope := sqrt(sqr(k2) + sqr(k3));

         pt(.r,c.).slope := slope;
         if (lam1>curv_thresh) and (lam2>curv_thresh) then
            pt(.r,c.).cur := 'd'     (* depression   *)
         else if (lam1<-curv_thresh) and (lam2<-curv_thresh) then
            pt(.r,c.).cur := 'h'     (* hill or peak *)
         else if (abs(lam1)<=curv_thresh) and
                 (abs(lam2)<=curv_thresh) then
            pt(.r,c.).cur := 'p'     (*    planar    *)
         else if (lam1<-curv_thresh) and (lam2>curv_thresh) then
            pt(.r,c.).cur := 's'     (*    saddle    *)
         else if (lam1<-curv_thresh) and
                 (abs(lam2)<=curv_thresh) then
            pt(.r,c.).cur := 'r'     (*    ridge     *)
         else if (abs(lam2)<=curv_thresh) and
                 (lam1>curv_thresh) then
            pt(.r,c.).cur := 'v'     (*    valley    *)
         else if (lam2<-curv_thresh) and (lam1>curv_thresh) then
            pt(.r,c.).cur := 'a'     (*    pass      *)
      end;
   end;


procedure smoothing(var points,spoints:point_array);
var
   r,c,i,j,temp,dif : integer;
begin

   for r := 2 to (row-1) do
   for c := 2 to (col-1) do
   begin
      if (points(.r,c.).slope > lobound)
      then begin
            temp := 0;
            for i := r-1 to r+1 do
               temp := points(.i,c.).el + temp;
            for j := c-1 to c+1 do
               temp := points(.r,j.).el + temp;
            spoints(.r,c.).el := trunc(temp/6);
      end
      else spoints(.r,c.).el := points(.r,c.).el ;
   end;
   for r := 1 to row do
   begin
      spoints(.r,1.).el := points(.r,1.).el;
      spoints(.r,col.).el := points(.r,col.).el;
   end;
   for c := 1 to col do
   begin
      spoints(.1,c.).el := points(.1,c.).el;
      spoints(.row,c.).el := points(.row,c.).el;
   end;
end;  (* procedure smoothing *)



procedure check(var pt:point_array);
var
   count,countb,countp,counth,countd :integer;
         counte,countr,countv,counta :integer;
   r,c,aa,cc : integer;
begin
   count :=0;
   countb :=0;
   countp :=0;
   counth :=0;
   countd :=0;
   counte :=0;
   countr :=0;
   countv :=0;
   counta :=0;
   aa := 0;
   cc := 0;
   for r := 1 to row do
   for c := 1 to col do
   if pt(.r,c.).slope < lobound then
      aa := aa + 1
   else if (pt(.r,c.).slope = 999.0) then
      countb := countb + 1
   else if (pt(.r,c.).slope > upbound) then
      cc := cc + 1
   else
   case pt(.r,c.).cur of
      'p' : countp := countp +1;
      'd' : countd := countd +1;
      'h' : counth := counth +1;
      'e' : counte := counte +1;
      'r' : countr := countr +1;
      'v' : countv := countv +1;
      'a' : counta := counta +1;
   end;
   writeln('level points are',aa);
   writeln('unsafe points are',cc);
   writeln('for safe points:');
   writeln('count planar =',countp);
   writeln('count hill =',counth);
   writeln('count depression =',countd);
   writeln('count etc    =',counte);
   writeln('count saddle =',counts);
   writeln('count ridge =',countr);
   writeln('count valley =',countv);
   writeln('count pass =',counta)*)
end;
```

```
(*=========================================================================
                                Main Program
 =========================================================================*)

begin
     reset(data,'data ell');
     rewrite(outslope,'data magl');
     rewrite(outcur,'data curl');
     rewrite(smoutel,'smdata ell');
     rewrite(smoutslope,'smdata magl');
     rewrite(smoutcur,'smdata curl');

     for c := 1 to col do
     for r := row downto 1 do
          readln(data,points(.r,c.).el);
      init(points);
      init(spoints);

      calc_slope_curvature(points);
      (* check(points) *)
      for c := 1 to col do
      for r := row downto 1 do
      begin
         writeln(outslope,points(.r,c.).slope);
         writeln(outcur,points(.r,c.).cur);
      end;
      for r:= 1 to 10 do begin
         smoothing(points,spoints);
         calc_slope_curvature(spoints);
         check(spoints);
         points := spoints;
      end;

      for c := 1 to col do
      for r := row downto 1 do
      begin
         writeln(smoutel,spoints(.r,c.).el);
         writeln(smoutslope,spoints(.r,c.).slope);
         writeln(smoutcur,spoints(.r,c.).cur);
      end;
end.
```

LIST OF REFERENCES

1. John C. Davis and Michael J. McCullagh, *Display and Analysis of Spatial Data*, John Wiley & Sons, 1975.

2. Theo Pavlidis, *Algorithms for Graphics and Image Processing*, Computer Science Press, 1987.

3. Dennis Duane Poulos, *Range Image Processing for Local Navigation of an Autonomous Land Vehicle*, Master's Thesis, Naval Postgraduate School, September 1986.

4. Donald Hearn and M. Pauline Baker, *Computer Graphics*, Prentice-Hall, 1986.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, CA 93943-5000	1
5.	US Army Combat Developments Experimentation Center (USACDEC) Attention: W.D. West Fort Ord, CA 93941	1
6.	Curriculum Office, Code 37 Computer Technology Naval Postgraduate School Monterey, CA 93943-5000	1
7.	Professor Neil C. Rowe, Code 52Rp Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
8.	Professor Robert B. McGhee, Code 52MZ Department of Computer Science Naval Postgraduate School Monterey, CA 93943	2
9.	Professor Michael J. Zyda, Code 52ZK Department of Computer Science Naval Postgraduate School Monterey, CA 93943	1

10.	Captain Do Kyeong Ok SMC 2211 Naval Postgraduate School Monterey, CA 93943	1
11.	Captain Seung Hee Yee Dongjak-gu Sadang-1dong 1019-10 Seoul Korea, 150	8